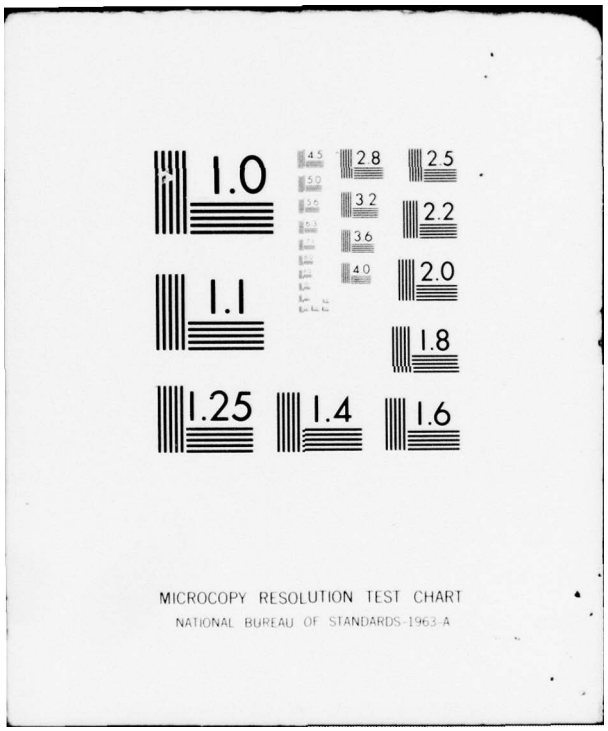
AD-A056 376 GENERAL ELECTRIC CO SCHENECTADY N Y RESEARCH AND DEV--ETC F/G 9/2
MADMAN ON THE H6000: DATA BASE MANAGER FOR HONEYWELL 6000. VOLU--ETC(U)
JUN 78 D E PHILLIPS, J M BROWN, J P KELLERMAN F30602-76-C-0321

UNCLASSIFIED RADC-TR-78-8-VOL-2 NL

1 OF 2
AD A056376


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

LEVEL III

A056375

(2)

SC

RADC-TR-78-8, Volume II (of two)
Final Technical Report
June 1978

MADMAN ON THE H6000
Data Base Manager for Honeywell 6000

D. E. D. Phillips
J. M. Brown
J. P. Kellerman

General Electric Company

Approved for public release; distribution unlimited.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-78-8, Volume II (of two) has been reviewed and is approved for publication.

APPROVED: Donald Van Alstine

DONALD VAN ALSTINE
Project Engineer

APPROVED: Wendall C Bauman

WENDALL C. BAUMAN, Colonel, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (ISIM) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: RADC-TR-78-8 Volume II (of two)	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): MADMAN ON THE H6000 Data Base Manager for Honeywell 6000 Volume II	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report 14 Jun 76 - 14 Jun 77
	6. PERFORMING ORG. REPORT NUMBER: N/A
7. AUTHOR(s): D. E. D. Phillips, J. M. Brown, J. P. Kellerman	8. CONTRACT OR GRANT NUMBER(s): F30602-76-C-0321
9. PERFORMING ORGANIZATION NAME AND ADDRESS: General Electric Company, Research and Development Center, 1 River Road Schenectady NY 12345	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62702F 55810269
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIM) Griffiss AFB NY 13441	12. REPORT DATE: June 1978
	13. NUMBER OF PAGES: 144
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Donald Van Alstine (ISIM)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

MADMAN	Hierarchical	Network Data Base
Data Base Management	CODASYL	
Command and Control	H6000	
Distributed Systems	PDP-11	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

MADMAN is a multi-access data base management system that exists on both the PDP-11 and the H6000. This report describes the version that exists on the H6000 under GCOS Versions 1/G, 2/H and 3/I.

MADMAN is implemented in such a manner that only one copy of the data base manager is resident on the system for each data base that has been opened. That copy can support a maximum of 16 application programs at one time. The application programs are written in FORTRAN using the CALL statement to access

DD FORM 1 JAN 73 1473

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

the data base manager. Data is transferred core-to-core between the application programs and the data base manager.

Three key features of MADMAN are:

1. Complete implementation of the CODASYL network data base standard (except for SELECT UNIQUE MASTER).

2. Sophisticated concurrency control with automatic rollback in case of data base conflicts.

3. Automatic rollbacks of processes that were active when systems crashed.

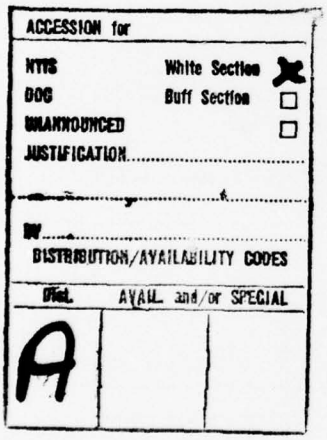


TABLE OF CONTENTS

PREFACE FOR VOLUME II

MADMAN is a multi-access data base management system that runs on the Honeywell 6000 series computer under the GCOS operating system, software releases 1/H, 2/G and 3/I. It provides efficient on-line concurrent access to network structured data stored on disc by FORTRAN programs.

This, the second volume of the final report on the MADMAN system, contains all the manuals associated with installing and running MADMAN on the H6000. The first volume describes MADMAN's design and features.

1. INTRODUCTION

MADMAN is a multi-access data base management system. Running as a free standing program, it provides easy access to a data base for a maximum of 16 FORTRAN application programs at one time. Communication between the application programs and MADMAN is performed by Intercom I/O, a standard GCOS III system routine.

MADMAN runs on any Honeywell 6000 series computer that supports the GCOS III operating system, software release 2/H. It is a complex system to install because of the interrelationship of files. On the other hand, it does not require any special hardware or any software changes to GCOS.

Figure 1 summarizes the tasks for installing MADMAN and the order in which they must be done. The people who perform the various tasks are represented by the columns. Tasks that can be done in parallel are shown in rows. Arrows indicate that some or all of the output of one task is input to another task. A '+' in the upper right corner of a box indicates that the task must be repeated for each data base. An absence of a '+' indicates that the activity need be done only once when the system is originally installed.

The people category consists of application programmers, Data Base Administrator and System Administrator. Only the tasks of the System Administrator and the Data Base Administrator are described in this manual. The assumption is made that both the System Administrator and the Data Base Administrator have a good working knowledge of the GCOS III operating system, GMAP and the control card language of GCOS.

This manual is written in a "cookbook" style, and is a step by step approach of how to install MADMAN on an H6000. No attempt has been made to explain the parts of MADMAN. The first section lists the abbreviations used in the manual and their definitions. The second section details the tasks for the System Administrator in the order they should be done. The third section details the tasks for the Data Base Administrator. The last section explains how to add a MADMAN data base management system for the second, third or nth data base.

ORDER OF TASKS FOR MADMAN

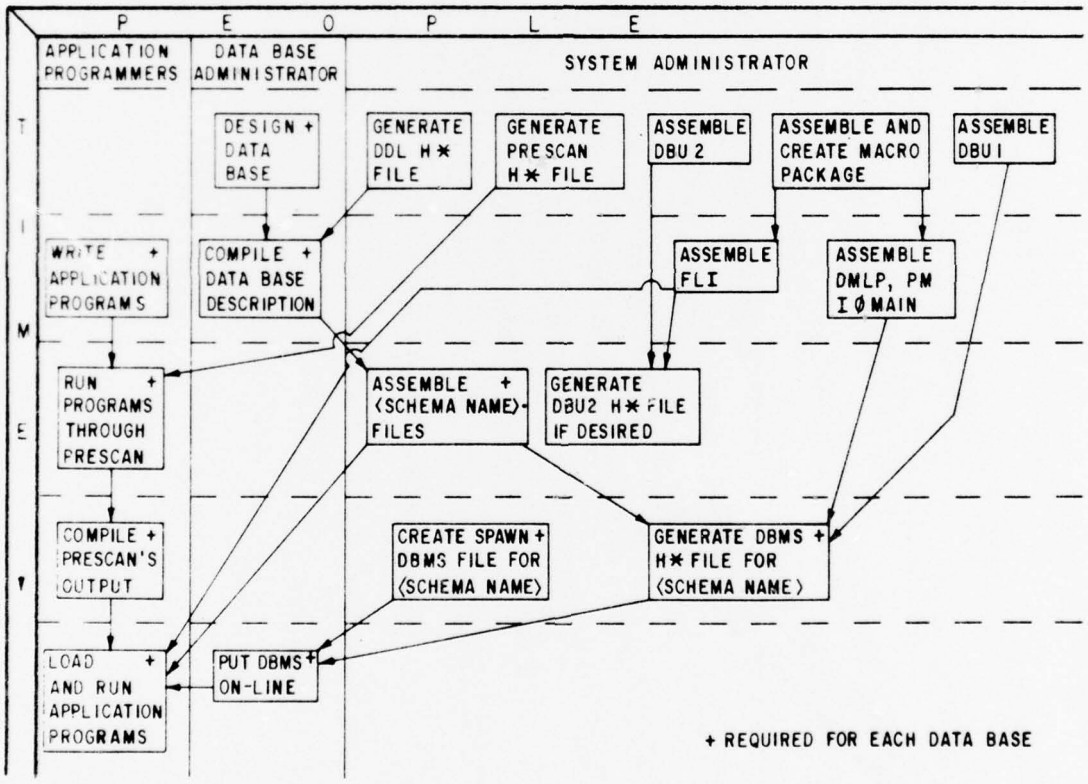


Figure 1

It is assumed that the batch jobs will be spawned under CARDIN. If the batch jobs are to be run from cards instead of CARDIN, the following control card must be added to each deck.

```
$       USERID  <userid$password>
```

Other available MADMAN manuals are:

(1) Data Definition Language Manual
(2) Data Manipulation Language Manual
(3) Data Base Utility One and Data Base Utility Two Manual
(4) Prescan Manual

2. ABBREVIATIONS

AP - application program, also referred to as user

DBMS - data base management system includes the DMLP, PM and DBU1

DBA - data base administrator

DBU1 - data base utility one

DBU2 - data base utility two

DDL - data definition language compiler

DMLP - data manipulation language processor, part of the DBMS

FLI - FORTRAN language interface

MADMAN - multi-access data base management system

PM - page manager, part of the DBMS

<schema name> - five characters or less that designate a particular data base

<userid> - user master catalog name for the Data Base Administrator or the System Administrator as the case may be

3. SYSTEM ADMINISTRATOR TASKS

The System Administrator has a fairly large number of tasks to perform when MADMAN is installed and the first data base is put on-line. After that, however, only the data sections need re-assembly and a spawn file added for each subsequent data base.

The System Administrator's tasks are described in the order in which they are to be performed except where indicated. If a task requires input from the Data Base Administrator, that is noted.

3.1 Intercom I/O

Before any data base can be put on-line, the GCOS Intercom I/O table must be expanded. The following control card in the GCOS system start-up deck accomplishes the expansion.

```
$       SCOMM   size,option
```

See the System Start-Up Manual (DD33), page 3-55 for details.

The algorithm for determining the maximum number of table entries for each data base is the maximum number of application programs that can talk to the data base at one time multiplied by three plus six or

size for one data base = 3 * max user + 6.

3.2 Macro Package

Before any of the data base management system (DBMS) source files can be assembled, the System Administrator must first assemble the macros and create a macro package. This step is a three activity job. The macros are assembled in the first activity; the last two activities make the macro package.

The control cards for making the macro package are:

```
$       IDENT   <userid>,MAD-MACS,ASSEMBLE MACROS & CREATE PKG
$       GMAP    DECK
$       LIMITS  07,55K
$       SELECTA <userid>/MAC/BGMACROS
$       SELECTA <userid>/MAC/FIELDS
$       SELECTA <userid>/MAC/MA12
$       SELECTA <userid>/MAC/MCC12
$       SELECTA <userid>/MAC/MCC34
$       SELECTA <userid>/MAC/MCC56
$       SELECTA <userid>/MAC/MCC7
$       SELECTA <userid>/MAC/MB1
$       SELECTA <userid>/MAC/MACS1
$       SELECTA <userid>/MAC/LODSTOR
$       SELECTA <userid>/MAC/CMPAR
$       SELECTA <userid>/MAC/ADDSUB
$       SELECTA <userid>/MAC/ANDMAC
$       SELECTA <userid>/MAC/QOR
$       SELECTA <userid>/MAC/CAND
$       SELECTA <userid>/MAC/ENMACROS
$       FILE    C*,X1S,20L,NEW,MADMACROS
$       TAPE    P*,X4D,,,,MADMACROS-P*
$       FILEDIT NOSOURCE,OBJECT,UPDATE
$       FILE    *R,X1R
$       FILE    R*,X2S,20L,NEW,MADMACED
$       DATA    *C,,COPY
$       INCLUDE
$       SYSLD   CATALOG=MADMC,ABSOL,MASTER
$       LOWLOAD
$       OPTION  NOGO,NOSETU
$       COPY    ,,MADMC
$       COPY    ,,MADMC
$       INCLUDE
$       EXECUTE
$       ENDLD
$       ENDEDIT
$       ENDCOPY
$       SYSEDIT
$       LIMITS  05,50K
$       FILE    R*,X2R
$       FILE    Q*,X3S,20R,NEW,MADMACROS
$       ENDJOB
```

If this job is being spawned from CARDIN, do a REMO CLEARFILES before the RUN; otherwise the AFT will overflow.

While the macro package is being made, the Data Base Utilities (DBU1 and DBU2), Prescan and Data Definition Language compiler (DDL) can also be installed.

3.3 Data Base Utilities

3.3.1 Data Base Utility One (DBU1)

At the same time the macro package is being created, DBU1 can be assembled because it does not require the macro package.

DBU1 is a section of DBMS. Since DBU1 is no longer needed once the data base is put on-line, it is loaded at the end of DBMS so that its core can be released once it has completed its work.

The control cards to assemble DBU1 are:

```
$       IDENT   <userid>,MAD-DBU1,ASSEMBLE DBU1
$       GMAP    DECK
$       SELECTA <userid>/DBU/DBU1MAC
$       SELECTA <userid>/DBU/DBU1DOCU
$       SELECTA <userid>/DBU/DBU1CON
$       SELECTA <userid>/DBU/DBU1
$       SELECTA <userid>/DBU/DBU1SUBS
$       SELECTA <userid>/DBU/DBU1INIT
$       SELECTA <userid>/DBU/DBU1REVE
$       SELECTA <userid>/DBU/DBU1STG
$       FILE    C*,X1R,10L,NEW,CSDBU1
$       ENDJOB
```

3.3.2 Data Base Utility Two (DBU2)

DBU2 can also be assembled in parallel with the macro package.

Unlike DBU1, DBU2 is a free standing program written in FORTRAN with two GMAP subroutines. In one sense, it can be considered as an application program because it uses the FORTRAN Language Interface (FLI) to communicate with a data base. On the other hand, it does not require the FLI storage associated with a particular data base in order to talk to that data base. Instead, DBU2 has its own special storage that allows it to perform the functions described in the Data Base Utility One and Data Base Utility Two manual.

DBU2 requires the object deck of the FLI before it can be run.

The control cards to compile DBU2 are:

```
$       IDENT   <userid>,MAD-DBU2,COMPILE DBU2
$       FORTY   DECK,NFORM,NLNO,BCD
$       FILE    C*,X1R,1L,NEW,CSDBU2
$       SELECTA <userid>/DBU/DBU2S
$       ENDJOB
```

The source for the FORTRAN and GMAP routines along with the control cards are in DBU2S. This job produces object files named CSDBU2, CSGEINFO and CSDBU.1.

3.3.3 Parameter File

Both DBU1 and DBU2 use the same file to maintain information for each data base. As each data base is initialized for the first time, its "vital statistics" are written into the file. The file must be created, however, by either the System Administrator or the Data Base Administrator. The filename is <userid>/DBPARMF. DBU1 and DBU2 can be made to reference another statistics file by using the PF command. See the Data Base Utility One and Data Base Utility Two manual for details.

3.4 Prescan

Prescan is another program that can be compiled and put in H* format for general use without the macro package. There is a version of Prescan that runs in batch and one that runs in time-sharing. The time-sharing version is distributed with MADMAN; the batch version is available upon request.

Prescan is a free standing program written in FORTRAN and contained in one file, SPRESCAN. To compile and install the time-sharing version in H* format, type OLD SPRESCAN and RUN under the YFOR system. The first line of SPRESCAN is a run command which stores the H* in a file named PRESCAN.

PRESCAN cannot be run until the schema description has been compiled by the DDL compiler because PRESCAN uses one of the files created by the DDL as input.

3.5 Data Definition Language Compiler (DDL)

The Data Definition Language compiler is supplied in H* format; therefore, there is no installation procedure.

3.6 Data Independent Assemblies

This section explains assembling those parts of MADMAN which require the macro package and which need to be assembled only once. These assemblies are referred to as data independent because they do not have to be repeated for each new data base. Included in this group are the source files for the Page Manager (PM), the Data Manipulation Language Processor (DMLP) and the FORTRAN Language Interface (FLI). All of these source files are essentially written in a macro language.

Except for the FLI, the resulting object files are used in every DBMS. The FLI's object file is loaded with every application program and DBU2; but, because it is independent of the schema description, it is assembled only once.

3.6.1 Page Manager (PM)

The control cards for assembling the Page Manager are:

```
$       IDENT   <userid>,*PAGE*,ASSEMBLE THE PAGE MANAGER
$       GMAP    DECK
$       LIMITS  50,59K,,50K
$       PRMFL   **,R,R,<userid>/MADMACROS
$       FILE    C*,X1R,5L,NEW,CSPAGMAN
$       TAPE    *1,X2R
$       SELECTA <userid>/DBMS/APPAGMAN
$       SELECTA <userid>/DBMS/MACEQUS
$       SELECTA <userid>/DBMS/EQUALS
$       SELECTA <userid>/DBMS/DATA
$       SELECTA <userid>/DBMS/PAGMAN
$       ENDJOB
```

The *1 file is put on tape because it grows to more than one million words.

3.6.2 Data Manipulation Language Processor (DMLP)

Ten separate assemblies are required to create the data independent object files for the DMLP. The control cards are the same for each assembly except for the names of two of the source files.

```
$       IDENT   <userid>,*<name>*,DMLP <name> SECTION
$       GMAP    DECK
$       LIMITS  <limits>,,40K
$       SELECTA <userid>/DBMS/BG<name>
$       SELECTA <userid>/DBMS/MACEQUS
$       SELECTA <userid>/DBMS/EQUALS
$       SELECTA <userid>/DBMS/DATA
$       SELECTA <userid>/DBMS/<name>
$       PRMFL   **,R,R,<userid>/MADMACROS
$       FILE    C*,X1D,10L,NEW,CS<name>
$       ENDJOB
```

where:

<userid> is the user master catalog
<name> is the name of the DMLP section
<limits> is the amount of time and core required for assembly

The values for <name> and <limits> are:

<name>	<limits>
IOMAIN	30,58K
MOVEIT	15,57K
STORE	24,57K
GET	15,57K
VFY	24,57K
FIND	24,57K
RMV	20,57K
FINDA	15,57K
FINDK	20,57K
CHGINC	15,57K

3.6.3 FORTRAN Language Interface (FLI)

The FLI does preliminary checking on the arguments given in the application program's calls to the DMLP and passes these arguments to the appropriate DBMS using Intercom I/O. The object file from this assembly must be loaded with each application program and also with DBU2.

The control cards for the FLI assembly are:

```
$       IDENT   <userid>,*FLI*,ASSEMBLE FLI
$       GMAP    DECK
$       LIMITS  05,57K,,40K
$       PRMFL   **,R,R,<userid>/MADMACROS
$       FILE    C*,X1D,10L,NEW,CSIOFLI
$       SELECTA <userid>/FLI/IOFLI
$       SELECTA <userid>/FLI/FLIERR
$       ENDJOB
```

Control cards in the source files cause the object file CSFLIERR to be created.

3.7 Data Dependent Assemblies

There are four assemblies of data source files required for each data base. These source files are generated by the DDL and describe the schema, the data base data files, the user's buffers for the DBMS, and the data base records for the FLI.

The assemblies of these source files cannot be done until the Data Base Administrator has designed the data base and its description has been compiled by the DDL.

The control cards to assemble the four storage files are as follows:

3.7.1 Schema Storage

This assembly uses the '.L' and '.O' files produced by the DDL compiler to generate the tables that describe the schema organization. Its control cards are:

```
$       IDENT   <userid>,<schema name>STG,DBMS SCHEMA STORAGE
$       GMAP    DECK
$       LIMITS  24,58K
$       PRMFL   **,R,R,<userid>/MADMACROS
$       FILE    C*,X1R,10L,NEW,<schema name>.SC
$       SELECTA <userid>/STG/APSCHSTG
$       SELECTA <userid>/DBMS/MACEQUS
$       SELECTA <userid>/DBMS/DEFS
$       SELECTA <userid>/<schema name>.L
$       SELECTA <userid>/STG/SCHDEF
$       SELECTA <userid>/<schema name>.O
        END
$       ENDJOB
```

3.7.2 Data Base Data Files

The data base data files describe the disc files that eventually will contain the data. The output of this assembly supplies the Page Manager with such information as the names of the data base storage files and the pages assigned to each area. The control cards for this assembly are:

```
$       IDENT   <userid>,<schema name>DBS,DB DATA FILES
$       GMAP    DECK
$       FILE    C*,X1R,1L,NEW,<schema name>.PI
$       SELECTA <userid>/STG/PIUT
$       SELECTA <userid>/<schema name>.A
        END
$       ENDJOB
```

3.7.3 Users' Buffer Storage

This assembly generates the buffers in the DBMS for the application programs. Sixteen is the maximum number of application programs that can be running against a data base at one time. Since approximately 2000 words are needed for each user's buffer, the System Administrator has been given the ability to change the maximum number of users a DBMS can accept. The variable NBUFS in file <userid>/USERBUFC at line 650 indicates the maximum number of users. NBUFS can be any value from 1 through 16. The results of changing NBUFS to any number outside this range are unpredictable. MADMAN is distributed with NBUFS set to 7.

Sample:

```
650:CONST:NBUFS,7:DEFINE THE MAXIMUM NUMBER OF BUFFERS
```

change the '7' to the desired value.

The control cards to assemble the users storage are:

```
$       IDENT   <userid>,<schema name>USR,DBMS USER STORAGE
$       GMAP    DECK
$       LIMITS  24,58K
$       PRMFL   **,R,R,<userid>/MADMACROS
$       FILE    C*,X1R,2L,NEW,<schema name>.US
$       SELECTA <userid>/STG/APUSRSTG
$       SELECTA <userid>/DBMS/MACEQUS
$       SELECTA <userid>/STG/USERBUFA
$       SELECTA <userid>/STG/WKSTR
$       SELECTA <userid>/STG/USERBUFB
$       SELECTA <userid>/<schema name>.L
$       SELECTA <userid>/STG/USERBUFC
$       ENDJOB
```

3.7.4 Record Description for FLI

The purpose of this assembly is to generate the tables that describe to the FLI the records of a particular data base. The object file from this assembly, along with the FLI object file, must be loaded with every application program that accesses the particular data base.

The control cards for this assembly are:

```
$       IDENT   <userid>,<schema name>FLI,RECORD DESC FOR FLI
$       GMAP    DECK
$       FILE    C*,X1R,1L,NEW,<schema name>.I
$       SELECTA <userid>/<schema name>.W
        END
$       ENDJOB
```

3.8 MADMAN DBMS H* File

At this point all the object files necessary to put together a DBMS for a data base have been made. To load all the object files and store the result in program link (H*) format, use the following control cards.

```
$       IDENT   <userid>,<schema name>,MADMAN DBMS
$       OPTION  NOGO,ERCNT/20/,SAVE/DBMS
$       LOWLOAD 36
$       USE     .RTYP
$       ENTRY   DBU1
$       SELECT <userid>/CSIOMAIN
$       SELECT <userid>/CSMOVEIT
$       SELECT <userid>/CSSTORE
$       SELECT <userid>/CSGET
$       SELECT <userid>/CSVFY
$       SELECT <userid>/CSFIND
$       SELECT <userid>/CSRMV
$       SELECT <userid>/CSFINDA
$       SELECT <userid>/CSFINDK
$       SELECT <userid>/CSCHGINC
$       SELECT <userid>/CSPAGMAN
$       SELECT <userid>/CS<schema name>.SC
$       SELECT <userid>/CS<schema name>.PI
$       SELECT <userid>/CS<schema name>.US
$       SELECT <userid>/CSDBU1
$       EXECUTE DUMP
$       LIMITS  02,<core>,-6K
$       FILE    H*,X1D,50R,NEW,HS<schema name>
$       ENDJOB
```

<core> varies depending upon the number of records and set types in the schema and the number of users' buffers. 49K is sufficient to create most MADMAN DBMS's.

3.9 Spawn File for MADMAN DBMS

In order to put a data base on-line (run the H* file) a set of control cards must be put in a spawn file under the userid OPNSUTIL. There must be a spawn file for each data base. The name of the spawn file and the snumb for the job are the schema name. Since the snumb is limited to five characters or less, all schema names must also be five characters or less.

The contents of each spawn file are:

```
$       IDENT   <userid>,<schema name>,PUT MADMAN DBMS ON-LINE
$       USERID  <userid$password>
$       LOWLOAD
$       OPTION  NOSETU

 MADMAN environment conditioning object deck

$       EXECUTE DUMP,NJREST
$       PRIVITY
$       LIMITS  10,<core>,,99999
$       SELECT  <userid>/<schema name>.SL
$       ENDJOB
```

Several parts of the spawn file need explanation. Probably most important is the MADMAN environment conditioning routine. Its purpose is:

(1) To establish certain system parameters
(2) To read in the H* file

The system parameters that are modified are the time limit and the I/O queues pointer and counter. To assure system security, the routine revokes privity after the changes have been made. The bits indicating privity are cleared in both the .STATE word and the table containing the program's attributes. As a further precaution, the routine also sets the .SNPAI to indicate that it is the last activity of the job.

For system security reasons, this routine is supplied to the System Administrator on cards and each data base must be spawned from the operator's console. (See the source listing of the routine for documentation.)

<core> varies depending upon the number of set types and record types in the data base and the maximum number of user buffers. The algorithm for <core> is:

```
17K  for MADMAN
 mK  for users buffers
 nK  for data base description storage
```

where m is the size of the user buffer storage (one user buffer takes approximately 2K words) and n is the size of the DBMS schema storage plus the Page Manager storage.

The <schema name>.SL file permits the Data Base Administrator to communicate with the DBMS. It is in BCD card format and it contains:

1. A control card indicating the H* file to be run.

```
$       PRMFL   H*,R,R,<userid>/HS<schema name>
```

2. The control cards and/or data cards for communication with DBU1.

The two file codes DBU1 uses for communication are IT for input and EM for output. There are several options open to the DBA for assigning EM and IT. One is to put DAC cards in the <schema name>.SL file for the two file codes. This option creates a pseudo time-sharing mode in which the DBA connects to the DBMS with a terminal and responds to the requests for input. The control cards for this option are:

```
$       DAC     05
$       DAC     06
```

Another option is to put the commands to DBU1 in the <schema name>.SL file. The control cards are:

```
$       SYSOUT  EM
$       DATA    IT

    commands to DBU1 (See Data Base Utility
    One and Data Base Utility Two manual)
```

A third option is to use a PRMFL card for IT.

If the LIMITS card in the spawn file is incorrect, it can be overwritten by putting a LIMITS card in the <schema name>.SL file.

4. DATA BASE ADMINISTRATOR TASKS

For each data base, there is a Data Base Administrator (DBA) who is responsible for designing the schema and describing it in such terms that the Data Definition Language compiler (DDL) can compile it. Another DBA task is to put the DBMS on-line and to take it down. The DBA is also responsible for explaining to the application programmers how to run Prescan and what control cards are necessary to run an application program (AP).

The following sections elaborate on the DBA's tasks.

4.1 Data Definition Language Compiler (DDL)

One of the first tasks of the System Administrator is to install the DDL compiler. The time-sharing DDL compiler is normally supplied with MADMAN on the H6000; however the batch version is available upon request. In either case the only input is the name of the file that contains the schema description. The file name must also be the data base name. Since the data base name is also used as a snumb, it and consequently the filename must be five characters or less. The DDL compiler reads either ASCII or BCD files; therefore, the schema description file can be built in time-sharing. See the DDL manual for details concerning schema description.

To run the time-sharing version of the DDL compiler, under the YFOR system type GET <userid>/TDDL6000,R"DDL" and then RUN DDL.

The control cards to run the batch DDL compiler are:

```
$       IDENT   <userid>,DDL-H6000,COMPILE H6000 DATA BASE
$       PROGRAM DDLCMP
$       LIMITS  10,50K
$       PRMFL   H*,R,R,<userid>/BDDL6000
$       PRMFL   **,R,R,<userid>/BDDL6000
<schema name>
$       ENDJOB
```

The output of the DDL compiler is six ASCII files, usually referred to as the dot (.) files. The <schema name>.P file is essentially a listing of the compilation. The <schema name>.L file, the <schema name>.O file and the <schema name>.A file are inputs to the storage assemblies for the DBMS. The <schema name>.W file is input to the storage assembly for the FORTRAN Language Interface (FLI). The <schema name>.F file is input to

the Prescan program. Thus the System Administrator cannot build a DBMS until the schema description is successfully compiled.

4.2 Running a MADMAN DBMS

Once the System Administrator has created a DBMS for a particular data base, the Data Base Administrator is responsible for putting it on-line and shutting it down. To put the data base on-line, the DBA

1. Puts the appropriate control cards in the <schema name>.SL file. This file is in BCD card format. It contains the PRMFL card indicating the DBMS H* file and the control cards and/or data cards for DBU1 input and output. (See System Administrator's Tasks, section IX and the Data Base Utility One and Data Base Utility Two manual).

2. Asks the computer operator to spawn MADMAN DBMS for the particular data base. The spawn file has the same name as the data base and the schema description file. This name becomes the snumb; therefore, it must be five characters or less.

3. If DAC is used to communicate with the DBU1, the DBA can now connect to the DBMS with a terminal.

To shut the data base down or to communicate with the DBMS after the data base is on-line, the DBA runs DBU2. The communication with DBU2 is accomplished through DAC. The DBA connects to DBU2 after DBU2 is initiated under CARDIN. Input and output is then performed in a pseudo time-sharing environment.

DBU2 can also be run strictly batch. See the Data Base Utility One and Data Base Utility Two manual for details.

The control cards to run DBU2 are:

```
$       IDENT   <userid>,**DBU2**, RUN MADMAN DBU2
$       OPTION  FORTRAN
$       USE     .RTYP
$       USE     .GTLIT
$       SELECT  <userid>/CSDBU2
$       SELECT  <userid>/CSGEINFO
```

```
$       SELECT  <userid>/CSDBU2.I
$       SELECT  <userid>/CSIOFLI
$       SELECT  <userid>/CSFLIERR
$       ENTRY   DBU2
$       EXECUTE DUMP
$       DAC     05
$       DAC     06
$       ENDJOB
```

See the Data Base Utility One and Data Base Utility Two manual for the commands available to DBU1 and DBU2.

4.3 Prescan

Prescan is available in both a batch and a time-sharing version. The time-sharing version is distributed with MADMAN; the batch version is available upon request.

Before Prescan can be run, the System Administrator must make the Prescan H* file and the Data Base Administrator must successfully compile the schema description. The input to Prescan is the <schema name>.F file produced by the DDL compiler and a FORTRAN application program (AP). The output is a FORTRAN program that contains the AP and labeled commons and variables related to the schema.

To run Prescan in time-sharing at the SYSTEM? Level, type /PRESCAN.

See the Prescan manual for details.

The control cards to compile the application program after it has been through Prescan are

```
$       IDENT    <userid>,<identification>
$       FORTY    DECK,ASCII,<other options>
$       SELECTA  <userid>/<filename>
$       FILE     C*,X1D,1L,NEW,<object filename>
$       ENDJOB
```

4.4 Running an Application Program

After the FORTRAN Language Interface (FLI) has been assembled, the FLI storage for the given data base has been assembled and the application program (AP) has been compiled, the AP can be run (assuming the DBMS is on-line).

The control cards to "load and go" are:

```
$       IDENT   <userid>,<identification>
$       LOWLOAD
$       OPTION  FORTRAN
$       SELECT  <userid>/<AP object file>
$       SELECT  <userid>/CSIOFLI
$       SELECT  <userid>/CSFLIERR
$       SELECT  <userid>/<schema name>.I
$       USE     .GTLIT
$       EXECUTE

   other control cards and/or data

$       ENDJOB
```

5. ADDING NEW MADMAN DBMS's

The mechanism that tailors a MADMAN DBMS to a particular data base is the tables created by the DDL compiler. For the second, third, nth DBMS, therefore, only the storage files need re-assembly. The System Administrator's tasks for adding a new DBMS once the first one is installed are:

1. Assemble the data dependent files
   (System Administrator Tasks, Section VII)

2. Make the DBMS H* file
   (System Administrator Tasks, Section VIII)

3. Create the spawn file
   (System Administrator Tasks, Section IX)

The System Administrator may also have to expand the inter-slave communication table.

The Data Base Administrator tasks, however, all derive their input from the description of the particular data base. Thus the DBA must repeat every task for each new data base.

MULTI-ACCESS DATA BASE MANAGEMENT SYSTEM

(MADMAN FOR THE PDP-11 AND THE H6000)

DATA DEFINITION LANGUAGE MANUAL

This manual contains the documentation for the Multi-Access Data Base Management System (MADMAN FOR THE H6000) Data Definition Language (DDL). The information contained in this document is not intended for a novice reader; the reader is expected to have some knowledge of data base facilities and languages to use those facilities.

MADMAN was originally implemented on the PDP-11 under the RSX-11D operating system. For the most part, the descriptions contained herein are machine and system independent. Some of the data formats, however, are based on a word size of two bytes (16 bits) for the PDP-11, 4 bytes (36 bits) for the H6000. An attempt has been made to modify the references to PDP-11 data formats to reflect those of the H6000.

January 1976 for MADMAN on the PDP-11
June 1977 for MADMAN on the H6000

1. LANGUAGE SPECIFICATIONS for DDL

This section contains language specifications for the Data Description Language.

1.1 Notation

1. A construct enclosed in square brackets ([]) is optional and may be omitted from the source schema. This notation is used to designate optional constructs whose presence or absence alters the meaning of the schema.

2. A construct enclosed by "{" and "}⊕" may be repeated in the source schema. If the braces are omitted, the "⊕" applies to the immediately preceding metavariable or optional construct.

3. If repetition of an optional construct is indicated, a minimum of zero occurrences is allowed.

4. Words in the source schema must be separated by a space character, comma, semicolon, carriage return, or any combination thereof.

5. Except as noted above, items enclosed by angular brackets (<>) denote variables for which appropriate values must be substituted in the source schema, and items not so enclosed are literals which must appear as given in the syntax. Underlined literals are mandatory, and those not underlined may be omitted without altering the meaning of the schema.

6. Exclamation points (!) are used to delimit comments in the source schema. All characters between and including paired exclamation points will be ignored by the compiler. Such comments may appear anywhere in the source schema that a space character may appear, and will be treated as a space.

1.2 DDL Syntax

<source schema> ::= <id section> [<access section>] <area section> <record section> <set section> END

<id section> ::= SCHEMA <schema name> VERSION <integer> DDL <release>

<schema name> ::= <identifier>

<release> ::= <integer>

<access section> ::= KEYS <key declaration list> [<key group>]@

<key declaration list> ::= <key> ¦ <key> <key declaration list>

<key> ::= <identifier>

<key group> ::= OR <key group name> EQUIV <key reference list>

<key group name> ::= <identifier>

<key reference list> ::= <krl element> ¦ <krl element> <key reference list>

<krl element> ::= <key> ¦ <key group name>

<area section> ::= <area declaration>@

<area declaration> ::= AREA <area name> IS <integer> PAGES IN FILE ' <file id>'

<area name> ::= <identifier>

<record section> ::= <record declaration>@

<record declaration> ::= <record id> [<item declaration>]@ ¦ RECORD NULL

<record id> ::= RECORD <record name> [<read lock>] [<write lock>] <storage strategy>

<record name> ::= <identifier>

<read lock> ::= READERS <key reference list>

<write lock> ::= WRITERS <key reference list>

<storage strategy> ::= STORE <mode>

<mode> ::= <unique mode> | <direct mode> | <interval mode> | <set mode>

<unique mode> ::= UNIQUE IN AREA <area name> ON PAGE <integer>

<direct mode> ::= DIRECT [IN <area specification>]

<area specification> ::= AREA <area list> | AREAS <area list>

<area list> ::= <area name> | <area name> <area list>

<interval mode> ::= INTERVAL <integer> IN <area specification>

<set mode> ::= <nonreserved set mode> | <reserved set mode> | <calc set mode>

<nonreserved set mode> ::= PER <set name> [THEN IN <area specification>]

<reserved set mode> ::= PER <set name> [PERMANENTLY] {RESERVING [EXCLUSIVE]
[<integer-1> TO] <integer-2> NEAR OWNER [IN <area specification-1>]
THEN [<adjustment>] [<integer-3> TO] <integer-4> IN <area specification-2>}@

<calc set mode> ::= PER CALC IN <area specification>

<adjustment> ::= IMMEDIATELY | ADJUST1 | ADJUST2

<item declaration> ::= ITEM <item name> IS <representation> [<occurrences>] [<use lock>]

<item name> ::= <identifier>

<representation> ::= <integer> BYTE <type>

<type> ::= INTEGER | REAL | CHARACTER

<occurrences> ::= OCCURS <integer>

<use lock> ::= USERS <key reference list>

<set section> ::= <set declaration>@

<set declaration> ::= <set attributes> <member declaration>@ | SET CALC <calc declaration>@ | SET NULL

<set attributes> ::= SET <set name> OWNER IS <record name> INSERT <insertion> [LINK TO <linkage>]

<set name> ::= <identifier>

<insertion> ::= FIRST | LAST | BEFORE | AFTER | SORTED | SORTED WITHIN TYPE | INDEXED

<linkage> ::= <link type> | <link type> <linkage>

<link type> ::= PRIOR | OWNER | LAST

<member declaration> ::= MEMBER IS <record name> [<membership>] [LINK TO OWNER]
[SORT ON <item list> DUPLICATES <dup control>]

<membership> ::= [MANDATORY | OPTIONAL | FIXED] AUTOMATIC | MANUAL

<item list> ::= <item name> [<direction>] | <item name> <direction> <item list>

<direction> ::= ASCENDING | DESCENDING

<dup control> ::= BEFORE | AFTER | REJECTED | ALLOWED

<calc declaration> ::= MEMBER IS <record name> CALC ON <item list> [USING <procedure name>]
DUPLICATES <dup control>

<procedure name> ::= <identifier>

1.3 Reserved Words

ADJUST1
ADJUST2
AFTER
ALLOWED
AREA
AREAS

ASCENDING
AUTOMATIC
BEFORE
BYTE
CALC
CHARACTER
DDL
DESCENDING
DIRECT
DUPLICATES
END
EQUIV
EXCLUSIVE
FILE
FIRST
FIXED
IMMEDIATELY
IN
INDEXED
INSERT
INTEGER
INTERVAL
IS
ITEM
KEYS
LAST
LINK
MANDATORY
MANUAL
MEMBER
NEAR
NULL
OCCURS
ON
OPTIONAL
OR
OWNER
PAGE
PAGES
PER
PERMANENTLY
PRIOR
READERS
REAL
RECORD
REJECTED
RESERVING
SCHEMA
SET
SORT

SORTED
STORE
THEN
TO
TYPE
UNIQUE
USERS
USING
VERSION
WITHIN
WRITERS

1.4 Semantics and Notes

GENERAL:

The use of any character not defined here will produce an error message and require that the line be re-input to the compiler from the point of error.

The value of <identifier> may never duplicate a reserved word.

DDL:

The DDL is the language for describing the structure of a data base. The description written in the DDL is the source schema. The source schema consists of five parts:

A schema id section which identifies the schema,
An optional access control section,
A storage area description section,
A record description section, and
A set description section.

ID SECTION:

<schema name> is the identifier used to reference the schema and distinguish it from other schemata. Schema names may be a maximum of five characters in length.

VERSION <integer> specifies the version or generation of schema having this name.

<release> specifies the version of the DDL in which the source schema is written. If the compiler is not capable of accepting the designated version of the language, an error message will be generated and compilation will not be attempted.

ACCESS SECTION:

The access section defines application program authority keys and defines symbolic names for groups of these keys. These specify the subsets of the data base that are visible to application programs. Applications programs specify the authority key or key group name in their INVOKE declarations and DBOPEN procedure calls.

<key declaration list> names application program authority keys. A maximum of 15 keys may be named.

<key group> defines and names groups of keys for convenient reference. <key group name> is the name given to the group of keys referenced in the corresponding <key reference list>arguments.

Appearance of a key group name in the source schema is equivalent to appearance of its associated key reference list. If key group names appear on a key reference list, this equivalence is applied recursively.

Up to 32000 key groups may be declared.

Any key may be mentioned any number of times in key reference lists.

Circular definition of key groups is not permitted.

If the access section is omitted, all programs are granted full access to all portions of the data base.

In any case, a special authority key DBAKEY available only to the data base administrator permits unrestricted access to the entire data base. This key need not be declared or included in any key reference lists.

If a key reference list consists of a single key name, then the associated key group name is a synonym for the key name.

Key group declarations are strictly for the convenience of the DBA to avoid the need for repeated occurrences of identical lengthy key lists in the source schema.

The ability to perform retrieval operations on sets requires read access to the owner record type as well as to any member types being accessed.

AREA SECTION:

The computer system is assumed to provide storage space composed of files, which in turn consist of an assigned number of pages of uniform size. The number of pages in a file is assigned by the appropriate system control facilities.

The area section of the DDL defines portions of files to be used for data storage.

Each area is contained in a single file and occupies a specified number of pages within the file. Any page of a file may be assigned at most to one area.

Each occurrence of <area declaration> specifies one area of the data base.

<area name> is the identifier by which the area is referenced elsewhere in the source schema and in the application program.

<file id> identifies the file within which the area is being defined. This must be in the form of an H6000 file designation: "/name".

<integer> specifies the number of pages from the file to be used for the area being specified. Areas are assigned within the file in the order in which their declarations are encountered.

The sum of all assignments specified by <integer> for any file may not exceed the number of pages available in the file.

A minimum of one area must be specified, and a maximum of 127 may be specified.

RECORD SECTION:

The purpose of the record section is to declare record types to be stored in the data base, component items of these record types, and the storage strategy for each such record type.

Each occurrence of <record declaration> declares one record type.

A minimum of one and maximum of 60 record types may be declared for MADMAN on the H6000. For MADMAN on the PDP-11, the maximum is 127.

<record name> is the identifier by which the record type is referenced in the application program and elsewhere in the source schema.

<read lock> specifies authority keys which permit a program to detect occurrences of records of this type. A record may be "detected" if it will satisfy a FIND operation regardless of whether or not the program has access to any data items within the record. This access is granted to a program if its authority key is contained either in <read lock> or <write lock> or in <use lock> for any component item of the record. Thus, <read lock> will normally specify only those authority keys for which programs are to be allowed to detect occurrences of records of this type, but for which no access is granted for any component item.

If any authority key appears in <write lock> or in <use lock> for any component item, its appearance in <read lock> for that record is optional and without additional effect.

<write lock> specifies authority keys which permit a program to modify, create, or destroy occurrences of records of this type. Modification includes altering item values and changing set participation.

A program whose authority key appears in <write lock> may initialize or modify only those items for which its authority key appears in <use lock>.

A special authority key DBAKEY available only to the DBA permits unrestricted access to the entire data base. This key need not be mentioned in the lock declarations.

If the ACCESS SECTION is omitted from the source schema, no locks may be declared. In this case, all programs have unrestricted access to the entire data base.

<storage strategy> specifies the algorithm to be used in locating storage space for each newly-created occurrence of the record type, and provides certain parameters to be used by the algorithm.

If UNIQUE mode is specified:

When the data base is initialized, a single occurrence of the record type is created and stored in the area and page specified. All items are initialized to binary zero. Additional record occurrences may not be created, nor may the single occurrence be destroyed.

UNIQUE records may not be members of any set.

During the lifetime of any run unit, the single occurrence of each UNIQUE record is always current of its type. It may or may not be available to the run unit, depending on authority key. A UNIQUE record may be accessed by name as current of its type without use of or reference to its data base key, set membership, or other conditions or criteria.

Except for the above restrictions, UNIQUE records may be declared and utilized in the same manner as any other record.

Any set owned by a UNIQUE record is termed a "singular" set. Every such set is always the current set of its type.

If DIRECT storage mode is specified:

The application program creating each occurrence of the record type must specify the area and page number where the record is to be stored. If no space is available on the designated page, an error is returned and the record is not created. The designated area must be one of the areas named by <area specification>.

If the optional <area specification> phrase is omitted, all areas of the data base are valid for storage of records of the type being declared.

If INTERVAL storage mode is specified:

The DBMS will maintain a "current area" designator for the record type. This will be global to all run units, and will point to one of the areas of <area specification>.

The DBMS will also maintain an "append pointer" for each area of the data base. These will be global to all run units, and each will point to one page within its area.

Each INTERVAL record is stored on the page designated by the append pointer of the current area for the record type. If sufficient space is not available on the page, the append pointer is incremented by one and the process repeated until space is found. If the end of the area is encountered, the append pointer is reset to the first of the area, the current area designator is changed to point to the next area specified by <area specification>, and the process repeated. If there are no other areas on <area specification>, the current area designator is reset to the first area of <area specification>.

Whenever the append pointer for an area is reset to the first of the area, all current area designators in the data base which are pointing to that area are changed to the next area of their respective area specifications. When an area specification is exhausted, the designator is reset to the first area.

Once space has been located for an INTERVAL record, the append pointer which was used is incremented by <integer> pages. If the end of an area is encountered, the area append pointer is reset, current area designators are changed as described above, and the new INTERVAL append pointer incremented by the number of pages by which the end of the previous area would have been exceeded. In other words, <area specification> is treated as a single, contiguous logical storage space.

If PER <set name> mode is specified:

<set name> is designated the prime storage set or simply the storage set for the record type being defined. Storage space for each record occurrence is located with respect to the occurrence of <set name> in which the record participates at the time it is created.

The record type being defined must be an automatic member of <set name>.

The nonreserved and reserved forms of PER <set name> control whether or not space will be reserved in advance

for storing groups of members on each storage set occurrence.

If the nonreserved form is used, no space is reserved in advance of storing occurrences of the record type being declared. When such a record occurrence is created, the DBMS attempts to place it at its insertion point on the storage set. That is, it attempts to find space on the page of the prior, next, or owner record of the storage set. There is no guarantee that storage will be attempted on all three of these pages, nor of the order in which these pages will be examined. If insufficient space is available on these pages, the areas named in <area specification> are searched until space is found. This search uses a current area pointer and area append pointers as though the record type required INTERVAL storage mode with an interval value of zero.

If the optional THEN phrase is omitted and the record cannot be placed at its insertion point, space is located by searching the area of the prior record from the insertion point. No current area pointer or area append pointer is used. Note that if all member types on a set use the nonreserved set mode omitting the THEN phrase, then members will always be stored in the area of the owner.

If the reserved set mode of storage strategy is used, then space may be reserved in advance for groups of members on the storage set. Blocks of reserved space are termed "partitions". Each partition is associated with exactly one storage set occurrence, and only eligible members of that set occurrence may be stored in the partition.

The DBMS never moves records between partitions. If a record is placed in a partition when it is created and later reassigned to a different occurrence of its storage set, it remains in its original partition as a "stranger" record.

Partitions for a record having reserved set mode may be created under two conditions: whenever a storage set owner is created or whenever an eligible record is to be stored and any existing partition has been filled. Partitions created with the owner record are placed on the page of the owner record and are termed "near" partitions. Other partitions are termed "overflow" partitions. Reserved space insures the clustering of records which might otherwise be scattered throughout the storage space.

Use of the optional term PERMANENTLY specifies "permanent" partitions. Otherwise, "disappearing" partitions are used. A permanent partition retains its identity as reserved storage space after being filled with member records. A

disappearing partition is destroyed once it is filled, leaving its component records stored as though they had never been placed in a partition. The option affects re-use of space vacated by deleted records and space overhead required to maintain partitions.

In either case, each storage set occurrence may have one partition identified as the active partition. This is the partition in which space is available for members to be stored. When the owner and near partition are created, the near partition is the active partition. If the set utilizes permanent partitions, then several may exist simultaneously for any set occurrence. However, only one is considered the active partition at any time. If disappearing partitions are specified, any set occurrence may have only one partition at a time, and that must be the active partition.

If a record is to be stored in permanent partitions, it will always be stored in a partition. This may be either the partition at the insertion point or the active partition. If the record type specifies disappearing partitions, and if there is no space in the appropriate partition on the page of the insertion point, then the DBMS will attempt to store the record on the page of the insertion point outside any partition. If this fails, the active partition will be used.

Only areas in which owners of <set name> may be stored may be named in <area specification-1>. Omission of <area specification-1> from the RESERVING phrase implies all such areas not explicitly named for the record type being declared. No area may be named more than once in the <area specification-1> phrases for any record type.

Whenever an owner of <set name> is stored in any area named or implied in <area specification-1>, the DBMS will attempt to reserve, on the same page, space for at least <integer-1> and no more than <integer-2> records of the type being defined. If the owner is being stored in a previously-reserved space, new space reservation is external to this but on the same page. If the owner being stored is eligible for placement in a previously-reserved space, it will be stored there even though space for <integer-1> member records cannot be reserved on the page. In this case, the largest possible space will be reserved.

In all other cases, the DBMS will insure that a near partition able to contain at least <integer-1> members will be created for each owner of <set name>. Since the owner of <set name> may itself have a storage mode of UNIQUE, DIRECT, etc., the space search for these modes will locate

a page with enough empty space to allow creation of the owner record and near partitions.

If the optional <integer-1> TO is omitted, then <integer-1> is assumed equal to <integer-2>.

<integer-1> may not exceed <integer-2>.

The total space required for an owner of <set name> and any near partitions may not exceed one page.

If the optional term EXCLUSIVE is used, the near partition will be available only to the record type being specified. Otherwise, it will be shared by all record types having reserved set storage mode, the same use of the PERMANENTLY option, and the same storage set owner. These other records may be other member types on the same set, or members of other set types.

If a partition may be shared by multiple record types, the minimum size will be the largest of the various minima, and the maximum will be the largest of the various maxima, converted to words of storage.

The THEN phrase specifies the manner in which overflow partitions are to be created. <integer-3> and <integer-4> specify the minimum and maximum number of records to be accommodated in each overflow partition. When an overflow partition must be created, the DBMS first tries the page of the insertion point. If there is not adequate space on this page, the areas specified in <area specification-2> are searched until a page with enough space is found. These areas are searched using area append pointers and a current area pointer, as in the nonreserved set mode.

<adjustment> specifies optional special treatment of the first overflow allocation. IMMEDIATELY specifies that only the specified near partition is to be placed on the page of the owner. Once the near partition is filled, no attempt is made to create an overflow partition or use "crack filling" on the page of the owner even though space may be available. The overflow areas are used as soon as the near partition capacity is exceeded.

ADJ1 specifies that the initial allocation plus first overflow allocation must equal at least <integer-2> records. ADJ2 specifies that the initial allocation plus first overflow allocation must equal at least <integer-2> plus <integer-3> records. All other overflow allocations are in the range <integer-3> to <integer-4> records.

A value of zero for <integer-2> indicates that no space is to be reserved when the owner is stored. "Crack-filling" will be attempted unless IMMEDIATELY is specified.

The PER <set name> options described above permit specification of near and near-star strategies in both competitive and reserved forms. Samples:

Near-competitive (IDS "near"):

```
... PER SLOPPYSET
```

Near-reserved:

```
... PER CLOSEBYSET RESERVING 5 TO 10 NEAR OWNER
THEN 10 TO 15 IN AREAS ALPHA, BETA
```

Near-star (immediate overflow reserved):

```
... PER FARAWAYSET RESERVING 0 NEAR OWNER THEN
IMMEDIATELY 15 TO 20 IN AREAS BLIGHTED
```

If PER CALC is specified, the CALC set is the storage set for the record type. <area specification> gives the areas over which record occurrences randomize. Each page of the data base contains a CALC set owner occurrence, and member storage is equivalent to the nonreserved set mode with no area specification.

If PER CALC is specified, the record type being specified must be named as a CALC set member in the set section of the schema.

<item declaration> defines component items of the record.

<item name> is the identifier by which the item is referenced elsewhere in the schema and in the application program.

<integer> defines the item size in bytes.

The value of <integer> must be valid for the mode specified:

INTEGER mode may specify 1- to 15-byte items
REAL mode may specify 4- or 8-byte items
CHARACTER mode may specify 1- to 127-byte (character) items

<occurrences> specifies the number of occurrences of the item in each record occurrence. If omitted, this is assumed equal to one. Multiply-occurring items are seen as one-dimensional arrays by the application program.

<use lock> specifies the authority keys which permit transfer of the item value to the UWA. If a run unit's authority key is contained in <key reference list>, the run unit will be allowed

to detect occurrences of the record type and the item value will be made available in the UWA by a GET or OBTN function referencing this record type.

If a run unit's authority key appears in both <use lock> for an item and <write lock> for the record type in which the item appears, then the run unit may modify values of the item.

A special authority key available only to the DBA permits unrestricted access to the entire data base. This key need not be specified in any lock declarations.

If the ACCESS SECTION is omitted from the source schema, no locks may be declared.

The RECORD NULL form of record specification causes the next sequential value of the internal record type designation to be assigned as an invalid value. Any DML function using an invalid type designation as an argument will return an error.

Values of internal record type designation are assigned sequentially as record declarations are encountered in the source schema. DML functions use internal type designation values as arguments.

SET SECTION:

Each occurrence of <set declaration> defines one set type. <set attributes> defines the set-level characteristics, and <member declaration> defines member-level characteristics for one or more member record types. SET CALC specifies the randomizing set whose attributes are system-defined, and <calc declaration> defines members of this set.

<set name> declares the identifier by which the set type is referenced elsewhere in the source schema and in the application program.

The compiler assigns type numbers to sets depending upon their order of declaration in the source schema. The reserved word NULL may be used for <set name> to suppress assignment of the next type number in the normal sequence. In this case, the remainder of <set declaration> is ignored and may be omitted. Such artificial set declarations have no effect other than to control type number assignment.

<record name> defines the owner record type for the set.

<insertion> specifies the ordering rules for the set:

FIRST or LAST: New members are inserted as first or last of member sequence, respectively, on selected set occurrence.

BEFORE or AFTER: New members are inserted before or after the 'current of set' of the selected set occurrence.

SORTED: Members are sorted on sort keys specified as part of the member declarations. Each set occurrence constitutes one sorted sequence of member records.

SORTED WITHIN TYPE: Occurrences of each member type on the set are sorted on sort keys specified as part of the member declarations. Otherwise, member sequence is determined in a manner convenient to the DBMS.

INDEXED: Set members are sorted as with the SORTED option, but in addition the system maintains an index allowing access to members for given sort key values without exhaustive searching of the set.

<link type> specifies optional pointers which may be used in addition to the required 'next' pointer for the set:

PRIOR: Each record contains a pointer to the previous record of the set (reverse set order).

OWNER: Each member contains a pointer to the owner of the set occurrence in which it participates.

LAST: The owner contains a pointer to the last member. If PRIOR or INSERT LAST is specified, such a pointer is present and LINK LAST is without additional effect.

Each occurrence of <member declaration> declares one member record type on the set being defined. <record name> is the record type being declared a member.

AUTOMATIC membership causes each record occurrence to be assigned to a set occurrence by the DBMS when the record is created.

MANUAL membership requires that the application program assign set membership by means of the INSERT procedure.

MANDATORY membership specifies that once an occurrence of the record type is made a member of any occurrence of the set type then it will always be a member of some occurrence of that set type. The record can be moved from one set occurrence to another by means of the CNGSET procedure.

OPTIONAL membership specifies that membership in an occurence of the set is not permanent and may be cancelled by means of the REMOVE procedure.

FIXED membership specifies that once an occurrence of the record type is made a member of any occurrence of the set type then it must remain a member of that occurrence of the set.

If the optional <membership> phrase is omitted, MANDATORY AUTOMATIC membership is assumed. If the <membership> phrase specifies only AUTOMATIC then MANDATORY AUTOMATIC membership is assumed. If the <membership> phrase specifies only MANUAL then OPTIONAL MANUAL membership is assumed.

SORT ON <item list> designates component items of the record which are to be concatenated to form the sort key for the record on <set name>. The SORT ON phrase must be used if the set is SORTED, SORTED WITHIN TYPE, or INDEXED, and may not be used otherwise. The major-to-minor sort sequence is the order in which items appear on <item list>. Any sort key component may be specified ascending or descending. If neither is specified, ascending is assumed.

<dup control> specifies action to be taken if duplicate combined sort keys are encountered on a single set occurrence. BEFORE and AFTER specify that a duplicate is to be placed immediately before or after previously-stored occurrences in the set sequence. REJECTED specifies that set assignment is not to be made, and that an error is to be returned to the application program. ALLOWED permits duplicates, with the order within any group of duplicates determined in a manner convenient to the DBMS.

<calc declaration> specifies members of the randomizing or calc set. <record name> is the record type being assigned membership on the set, and <item list> specifies items used for set selection and for sort keys. Each such item must be declared as an item in <record name>, and <record name> must specify the PER CALC storage strategy.

<procedure name> designates the randomizing procedure to be applied to <item list>. If this is omitted, a standard system procedure is used.

The calc set is SORTED WITHIN TYPE and linked next.

The SET NULL form of set specification causes the next sequential value of the internal set type designation to be assigned as an invalid value. Any DML function using an invalid type designation as an argument will return an error.

Values of internal record type designation are assigned sequentially as set declarations are encountered in the source schema. DML functions use internal type designation values as arguments.

MULTI-ACCESS DATA BASE MANAGEMENT SYSTEM (MADMAN FOR THE PDP-11 AND THE H6000)

DATA MANIPULATION LANGUAGE MANUAL

This manual contains the documentation for the Multi-Access Data Base Management System (MADMAN) Data Manipulation Language (DML). The information contained in this document is not intended for a novice reader; the reader is expected to have some knowledge of data base facilities and languages to use those facilities. He is also expected to have a working knowledge of FORTRAN IV.

MADMAN was originally implemented on the PDP-11 under the RSX-11D operating system. For the most part, the descriptions contained herein are machine and system independent. However, some of the data formats are based on a word size of two bytes (16 bits) for the PDP-11, 4 bytes (36 bits) for the H6000. An attempt has been made to modify the references to PDP-11 data formats to reflect those of the H6000.

Except for interfacing with the particular operating system, MADMAN on the PDP-11 and MADMAN on the H6000 are the same.

January 1976 for MADMAN on the PDP-11
June 1977 for MADMAN on the H6000

1. GENERAL DESCRIPTION OF MADMAN

1.1 Functions

The MADMAN data base management system (DBMS) provides an orderly and controlled means for a multiplicity of application programs written in Fortran to simultaneously access and update a data base.

The DBMS maintains stored data according to a predefined structure, and stores and retrieves data at the request of the application programs. The data is organized and managed by the DBMS so as to provide efficient access to desired subsets of the data, and to relieve the application program of details of data access procedures.

A Data Definition Language (DDL) is provided to allow the data base administrator to specify how the data is to be structured in the data base. The specification of the source schema is written in the DDL and is then compiled to produce an object schema or structure table for the data base. The object schema is loaded with the DBMS, so that it can properly manage the data base. An authority code for each record type and item in the data base is used to provide a subsetting mechanism.

A Data Manipulation Language (DML) is provided, in the form of a set of Fortran subroutines, to allow application programs to access and update the data in the data base. A prescan program is provided which scans the Fortran application programs and produces additional data declarations corresponding to the data items of the records accessed by the program. An application program declares which schema and subset of that schema it needs via an INVOKE statement. The prescan program uses the INVOKE statement and output from the DDL compiler to put the Fortran declarations in the application program.

The DBMS provides a means for the detection of conflicting accesses to the data base. Concurrency control is provided at the page level. A program wishing to modify a page will be allowed to do so only if no other programs are using it for reading or updating. Once a program has modified a page, the page is locked to others until the modifying program terminates. Multiple readers are allowed for a page, but no program can modify it until the readers have terminated. Resolution of conflicting accesses is invisible to the application programs.

The DBMS concurrency control mechanism is such that it differentiates between deadlocks and simple conflicts (conflicts which are not deadlocks). Simple conflicts are resolved by making an application program wait until the desired page becomes free. However, to resolve a deadlock, one or more of the conflicting programs is aborted and must be rerun.

In the event that a program is aborted, the DBMS rolls back the changes made to the data base by that program. This rollback is transparent to the application program.

A recovery capability is provided so that after a system failure changes can be rolled back for programs in execution when the system failed.

A Data Base Utility (DBU) is used to initialize a data base, put a data base on-line, and take a data base off-line. The DBU also provides other utility type functions, such as listing all data bases which are on-line. For MADMAN on the H6000, DBU is divided into two programs-- DBU1 which is loaded with the DBMS, and DBU2 which is a free standing program. See the Data Base Utility One and Data Base Utility Two manual for details.

1.2 Interfaces

Pictorially, the module interfaces between the parts of the DBMS is as shown in figure 1. The DBMS has interfaces to the application programs and mass storage device containing the data base and rollback file.

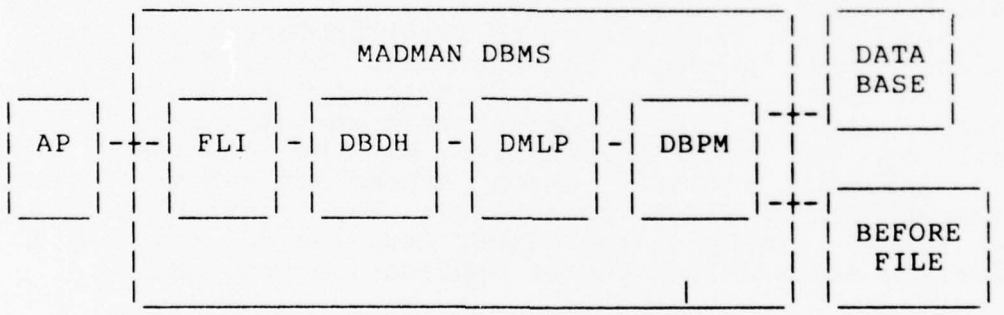


Figure 1

Application Programs (AP) interface to the DBMS via a set of Fortran subroutines called the Fortran Language Interface (FLI). The Fortran subroutine calls form the Data Manipulation Language (DML), which is documented below.

The mass storage device interfaces are input and output commands to the data base and before-page-image storage (also called the before file) for rollbacks.

1.3 Processing of a Typical Program on the PDP-11

To better illustrate the interfaces among the parts of the DBMS and between the DBMS and other parts of the system, we describe the path of flow for a typical program:

After an application program is placed into execution, as part of its initialization, it tells the DBMS of its intention to access a data base via a call to the DBOPEN procedure. The DBOPEN routine reformats the procedure parameters and passes the information to the Data Base Device Handler (DBDH). Based on this information and other default information, the DBDH allocates a suitable Data Manipulation Language Processor (DMLP) for the AP and supervises a check for other resources.

During the execution of the AP, calls are made on the DBMS via Fortran procedure calls. These calls transfer control to subroutines called the Fortran Language Interface (FLI). The FLI routines reformat the procedure arguments and provide some minor preliminary error checking and then pass the procedure parameters on to the DBDH via a system directive. The DBDH moves the arguments and any other data needed to process the procedure call into the addressing space of the DMLP corresponding to the AP. For protection reasons, this addressing space is entirely different from the AP addressing space.

The DMLP is now awakened and proceeds to interpret the procedure arguments and perform the desired actions. During this execution, the DMLP may call on the Data Base Page Manager (DBPM) to read or write pages of data from or to the data base. The DBPM handles all concurrent access control to the data base. For example, it does not allow two DMLP's, corresponding to two separate AP's, to have the same page of the data base at the same time for modification.

Also during execution of the DML procedure call, the DMLP may call on the DBDH to move data from the DMLP page buffers into the AP working storage area.

Upon completing the execution of the DML procedure, the DMLP returns status indicators to the DBDH, to be passed to the FLI and the AP. After checking for any error status, the FLI returns control to the user program.

Finally, as part of terminating, the AP issues a call to the DBTERM procedure to tell the DBMS that it is through using the data base and that the resources can be released.

During the execution of the AP, the DBMS keeps copies of the original contents of all data base pages modified by the AP. The original contents of these pages are saved on the 'before' file. In the event the AP is aborted, all the changes made to

the data base can be undone via the information in the before file.

Note that one of the main functions of the DBDH is to act as a communication interface between the AP and DMLP addressing spaces. Some operating systems have the facilities for interprocess communication required by MADMAN. On such systems, the other functions of the DBDH are absorbed by the FLI and DMLP and the DBDH module does not exist. On the H6000, the standard GCOS III system routine, Intercom I/O, takes the place of the DBDH.

1.4 Data Structures Supported

Data structures supported by the DBMS are a subset of the features described in the CODASYL Data Description Language Journal of Development [1]. This subset is similar to that used by IDS. CODASYL terminology is used in this document. However in this section, equivalent IDS features with different names are denoted by [IDS: <term>].

The structure of any data base supported by this DBMS must be predefined by a schema. The schema is written in the Data Description Language (DDL) by the Data Base Administrator (DBA), who is responsible for data base design and maintenance.

Elements of data structures are items [IDS: fields], records, and sets [IDS: chains]. Records are composed of items. Sets provide a mechanism for relating records which have some common significance but not necessarily common data.

The schema also specifies areas [IDS: subfile] or subdivisions of physical storage space in which records of each type will be stored. A record type may be restricted to certain areas.

The schema defines types of elements. Run units [IDS: programs] cause occurrences of these elements to be stored or accessed. It is important to distinguish between type and occurrence. The schema is prepared and maintained separately from both the data base and run units. There is no subschema as such, however a subsetting facility is provided, via authority codes.

Item types may be binary integer, binary floating point (long or short), or ASCII. Variable-length item types are not supported. Record types consist of a fixed number of occurrences of each of zero or more item types. Each item type must participate in exactly one record type.

For each set type, one record type must be designated the owner type and one or more other record types designated member types. Cycles are not supported. Otherwise, record types may

participate as owner or member in any number of set types. Records can become members of an occurrence of a set either automatically (when the record is created) or manually. Set membership can be optional, mandatory in a set but allowing set reassignment, or fixed in a particular set occurrence. Dynamic sets are not supported.

Set membership can be assigned in two ways: (1) automatically when a record occurrence is created or (2) manually by the application program.

Singular sets are supported. Associative access to any record type may be specified, using a randomizing (calc) procedure. User defined randomizing procedures are supported and different randomizing procedures can be specified for individual record types. Duplicate key values may be permitted in either case.

1.5 Data Manipulation Provisions

The Data Manipulation Language (DML) is the interface by which the run unit accesses the data base. The DML is similar to the proposed CODASYL facility [2], and consists here of a number of Fortran subroutine calls available to the run unit.

Data transfer between the data base and run unit is controlled by the DBMS, which copies data between a user working area (UWA) accessible to the run unit and the data base. The UWA consists of labeled common areas, one for each record type, the declarations for which are prepared by the prescan program.

Each record in the data base has a data base key (DBK) [IDS: reference code] which uniquely identifies the record to the DBMS and permits its retrieval. DBK values are invariant for the duration of run unit execution, but may be reassigned from one execution to another, due to data base restructuring or repacking.

The DBMS maintains currency tables for each run unit in execution. These tables contain the DBK of the current record of the run unit, current record of each record type, and current occurrence and record of each set type. Currency table updating may be inhibited by the run unit.

The process of locating a record in the data base is considered logically distinct from the process of moving data from the record to the UWA. The DML provides these separate functions as well as combined operations.

The DML functions may be divided into three categories: those dealing with data retrieval, those concerned with updating the data base, and miscellaneous operations. The last category

is concerned with control, status, and DBK manipulation. Procedures available in each category are described below.

Retrieval functions cause the DBMS to locate records and optionally copy them to the UWA. Any of the following may be used as the basis for locating records:

1) The DBK is specified by the run unit.

2) A key item value is specified by the run unit, and the record type was declared in the schema to be accessible in this manner.

3) The record participates in a set occurrence specified by the run unit. Set ordering characteristics are specified in the schema, and member records are accessed in this sequence.

4) The record is located in an area specified by the run unit.

In all but the first case above, there will generally be multiple records satisfying the specified condition. The DBMS will access and return these one at a time as requested by the run unit.

Data base update functions allow the run unit to create, destroy, and modify records and to assign and alter set participation. When a record is created or modified, appropriate data values are copied from the UWA to items in the record.

Whenever a record is to be made a member of a set, it is assigned to the current occurrence of the appropriate set type as indicated by the currency tables. The run unit is responsible for establishing the appropriate set currencies before issuing a procedure call which results in set assignment or reassignment. The DBMS does not support a set selection [IDS: unique master] capability which would locate appropriate sets based on values in the UWA.

The miscellaneous functions permit DBK values to be copied from the currency tables to run unit-addressable storage, permit testing for set membership assignment, and permit the run unit to specify error processing routines.

1.6 Authority Control

The principle of authority control is used as a method of implementing a subsetting mechanism and as a method to afford a measure of data protection at both the record and item level.

An authority code is associated with each record type, each item type within records, and each set type. The authority code indicates which run units have access to the records, items, or sets. The method of implementation allows for up to some maximum number of basic subsets. This maximum number of subsets is currently set to 15. Any program can be given access to one or more of these subsets. Each authority code consists of a bit mask, one bit for each subset. One bit is reserved as a special indicator for data base administrative functions.

When an application program opens the data base, it specifies the authority key with which it wants to access the data base. Any time the program requests access to a record, item, or set, its key is anded to the authority code to verify that the access can be granted.

Each record has both a read and a write authority code. Each item has only one authority code. These authority codes are specified in the DDL source schema. If an application is granted read or write access at the record level, it has the corresponding access to items in that record to which it has authority to access.

The set access is derived from the record accesses. An application program is granted access to a set type if it has access to the owner record type and at least one of the member record types.

By the use of the above procedure, an application program is given access to the data records, items, and sets which are pertinent to its operation, the remaining records, items, and sets being invisible to it. For example, if items are added to or removed from a record to which an application program has access and these are items to which the program does not have access, no changes are necessary in the application program for it to continue functioning properly.

References:

1. CODASYL Data Description Language Journal of Development. June 1973. National Bureau of Standards. NBS Handbook 113.

2. CODASYL COBOL Journal of Development - 1975. available from:
Technical Services Branch
Department of Supply and Services
5th Floor, 88 Metcalf Street
Ottawa, Ontario, CANADA
K1A, OS5

2. INTRODUCTION TO THE DATA MANIPULATION LANGUAGE

2.1 General Information

This section provides an introduction to the Data Manipulation Language (DML) and its use. The next section describes in detail each DML procedure available to the application programmer.

2.1.1 Application Programmer Level of Knowledge

The application programmer using this data manipulation language is expected to have a good working knowledge of the concepts and features of this data base facility. In addition, he is expected to understand the portion of the data structure pertinent to his program. For example, this level of knowledge is generally equivalent -- although not identical -- to that required for application programming using IDS.

2.1.2 Visible Data Structures

DBMS applications programs contain references to names of data base structures such as record types, data items, set types and areas. To execute correctly, application programs must contain declaration statements defining these names. Since the application programmer ordinarily does not have sufficient information about the structure of the data base to code these statements, the prescan preprocessor is used to generate and insert them into the application program prior to compilation. The application programmer tells prescan where to insert these declaration statements by placing a pseudo statement (INVOKE) in the source program.

These inserted statements declare all necessary symbolic names for item, record, and set types, for areas, and for DBMS parameters. They also create the User Working Area (UWA) in the form of labeled common areas within the run unit addressing space. Program references to data items are references to UWA locations. All other symbolic names are equated to numeric values used as identifiers by the DBMS. The programmer should always use symbolic names and generally need not be concerned with the corresponding numeric values.

Only the element types declared by these inserted declaration statements will be accessible to the run unit. Because of subsetting, entire records or items within records may be omitted from the declarations. In addition, a run unit may be allowed to modify some of the record types visible to it, but only read other types. The UWA is also accessible to the DBMS so that it may copy data between the data base and the UWA.

2.1.3 Data Base Keys

A record's data base key (DBK) is the logical address of the record in the data base. DBKs are 4 bytes in length, and contain the area identification, page number within the area, and logical record number (line number) within the page on which the record is stored. (The line number is in the first byte, area number in the second byte, and page number in the last two bytes on the H6000.)

A record may be physically moved within a page during space reclamation within the page. This does not alter its DBK.

If a record is physically moved from one page to another, its DBK will change. This never occurs as a result of DML calls by run units, and never occurs for any record during the lifetime of a run unit accessing that record. However, the data base administrator may cause certain record types to be repacked in storage to maintain operating efficiency. The DBKs of such records are subject to change between run unit executions, and should not be saved beyond the lifetime of the run unit.

2.1.4 Currency Tables

The DBMS maintains tables of DBKs of certain records processed by the run unit. These DBKs establish reference positions in the data base and permit efficient retrieval of the corresponding records. The DBMS retains (in these tables) the DBK of one record of each type. Each such record is termed current of type for its type. The DBMS also retains the DBK of one record in each area of the data base. Each such record is termed current of area for its area. The DBMS also retains the DBK of the record most recently processed by the run unit. This record is termed the current of run unit.

In addition to the above, for each set type, the DBMS maintains the DBKs of four records belonging to a single occurrence of that set type:

1. The record considered to be current of set,
2. The record following the current of set in set order,
3. The record preceding the current of set in set order,
4. The owner record of the set occurrence.

If the proper record corresponding to any currency table entry is not known to the DBMS at any time, a null value is used. For any set type, a current set is identified if any of the four set currency entries is non-null. In particular, a current set of a given type may be identified even though there is no record known as current of set.

Many DML functions utilize records identified by the currency tables. The programmer is responsible for establishing proper currency conditions before calling such functions. Generally, when a record is retrieved, created, modified, or otherwise processed as a result of a DML call, it becomes the current of run unit, current of its record type, and current of set for each set type in which it participates as owner or member. The run unit may optionally suppress the record and set currency updating. Updating of current of run unit can not be suppressed. Note that the currency tables can only be modified by the DBMS according to the result of a DML procedure. They cannot be directly modified by the run unit.

Indicators for current record of area are normally NOT updated (except for the FINDA and OBTNA procedures), but can be updated at the option of the run unit. When this updating is permitted, the record processed becomes the current record of the area in which it is stored.

2.1.5 DML Procedures

The following provides a brief description of the purpose of each DML procedure:

Data Retrieval:

Four procedures are available for locating a record which satisfies a specified condition. These procedures are FIND, FINDK, FINDD, and FINDA. They cause records to be located (DBKs found), but do not copy data from the data base into the UWA. There are four corresponding OBTN (obtain) procedures which locate records in the same manner as the FIND procedures and also copy data from the record into the UWA. Finally, the GET procedure copies data from the current record of the run unit into the UWA. A brief description of each retrieval procedure follows:

FIND and OBTN locate either the current record of a specified record type, or the current, next (in set order), prior, first, or last member record or the owner record of the current occurrence of a specified set type.

FINDD and OBTND locate a record whose data base key is specified by the run unit.

FINDK and OBTNK locate a record of the specified type whose key item value(s) is(are) specified by the run unit. The record type must have been declared in the schema to be accessible in this manner.

FINDA and OBTNA locate the current, next (in DBK order), prior, first, or last record within a specified area.

GET copies all or specified items from the current record of the run unit to the UWA.

Data Base Updating:

Nine procedures are available for updating the data base. These are STORE, STORED, MODIFY, INCR, CHANGE, CNGSET, INSERT, REMOVE, and DELETE.

STORE and STORED create a new record of the specified type, copy data values from the UWA to the record, and assign the record to set occurrences indicated by the currency tables. STORED is used to store records with a location mode of direct.

MODIFY, INCR (Increment), and CHANGE alter data items within the current record of the run unit. MODIFY causes all or specified items to be replaced by values from the UWA. INCR causes specified signed increments to be added to specified integer items within the record. CHANGE causes specified values to replace specified items within the record.

CNGSET, INSERT, and REMOVE modify set assignment of the current record of the run unit. CNGSET reassigns membership for mandatory or optional set members. REMOVE and INSERT are used respectively to remove optional members from set participation and to assign "ownerless" optional members to a set. INSERT is also used for assigning manual mandatory and manual fixed members to an occurrence of a set type.

DELETE destroys the current record of the run unit. Optionally, it may also destroy records which are members of sets for which the current record is the owner.

Miscellaneous Functions:

The miscellaneous operations deal with opening and closing the data base, status and control information, and data base key transfer.

DBOPEN prepares the data base for processing; DBTERM terminates the use of the data base; DBCLN declares a clean point during the processing; and DBERR sets the global error return.

MTSET and MEMSET are functions for determining whether a specified set occurrence has no members, or whether a specified record is presently a member of some occurrence of a specified set type.

MOVEC, MOVER, MOVES, MOVET, and MOVEA cause currency table entries to be copied from the DBMS tables into locations specified by the run unit. These procedures copy, respectively, the DBKs of current of run unit, current record of specified

record type, current record of specified set type, all four entries for a specified set type, and current record of a specified area.

2.1.6 Procedure Arguments

Arguments common to several DML calls are described below:

'record-name' is the symbolic name of a record type declared in the schema.

'set-name' is the symbolic name of a set type declared in the schema.

'item' is the name of an item declared in the schema.

'error-label' is an optional error exit designation, consisting of two parameters, the symbolic designator ERR followed by an integer variable to which a statement number has been assigned. For Fortran-Y, the form of the two parameters must be the symbolic designator ERL followed by the statement label or variable to which the label has been assigned preceded by a $, i.e., ERL,$40 or ERL,$L.

'area-id' is the symbolic name in an area in the data base.

'currency-option' specifies optional currency update suppression. These may include the literals RECORD, AREA, or SETS or up to five set type names. Area currencies are not updated unless specified in 'currency-option' (with the exception of FINDA and OBTNA).

'data-base-key' references a four-byte variable containing a DBK value.

'identifier' references a four-byte variable in which the DBMS will place a desired DBK value.

Arguments which apply to only a single DML call are explained with the call in the following section. The possible values of some of the arguments are specified as upper case words, such as ALL, SIMPLE, or NULL. These words are reserved; they may not be used as variable names, item names, or record or set types. If its value is desired, the word should be specified as the argument value. For example,

```
CALL GET(ALL)
```

2.1.7 Data Structure Definition Statements

Each application program needs the proper declaration statements describing the record and set types in the data base to which it has access. The prescan processor scans the application program and generates in the application program the necessary declarations so that the application program can access and update a MADMAN data base. These statements describe the records and items within those records to which the program has access. The statements also define symbolically the names for record types, set types, and other symbolic names such as ALL, NULL, NXTDUP, etc. In addition, one labeled common block is defined for communicating DBMS status returns.

2.1.7.1 The Invoke Statement

The INVOKE statement specifies the schema and subschema needed by the application program to access the data base. The position of the INVOKE statement in the source program specifies where the data base declaration statements are to be placed. The format of the INVOKE statement is as follows:

INVOKE [SUBSCHEMA <access key> OF] SCHEMA <schema name>

<access key>, if specified, must be a legitimate access key or group name defined in <schema name>. Prescan will replace the INVOKE statement with all record, set and data item declarations for <schema name> for which the associated access lock is matched by <access key>. The word INVOKE many begin in any column position but must be the first non-blank symbol on the line.

If <access key> is omitted and <schema name> contains no access section, all records, sets and data items from <schema name> will be declared. If <access key> is omitted but <schema name> does contain an access section, no declarations will be included in the application program.

Note that the PDP-11 Fortran compilers require that DATA statements follow all other declaration statements. Therefore the INVOKE statement must follow the last declaration statement in the application program and precede the first DATA statement.

2.1.7.2 Record Format Declarations

Each record format is described by a labeled common specification. The items within the record are the names of the variable locations defined within that labeled common. For example,

```
INTEGER PLATO,ARCHEM,ATOSSA,DARIUS
LOGICAL *1 ARSTOT
COMMON/XERXES/PLATO,ARCHEM,ARSTOT(10),DARIUS(6),ATOSSA
```

would define the layout visible to this user program for record type XERXES as 2 integers (1 word each), followed by a 10 byte item, followed by an integer item occurring 6 times, followed by an integer item. The item names, of course, are the variable names defined in the labeled common. These statements are the PDP-11 Fortran version; the Fortran Y version would be slightly different. (See the PRESCAN manual for MADMAN on the H6000.)

These labeled common blocks constitute the user work area (UWA). Prescan will only generate declarations for items and record types to which the application program has access.

It should be noted that the application program can refer to elements of an item type defined with the 'occurs' clause, but cannot refer to the entire group when performing DML procedures, unless the reference is to the entire record. In particular, this applies to the GET and MODIFY procedures.

2.1.7.3 Record and Set Type Declarations

Each set type and record type to which the application program has access is defined using either the DATA and INTEGER statements (for PDP-11 Fortran) or the PARAMETER statement (for Fortran Y). For example,

```
INTEGER XERXES,GREECE,PERSIA,TURKEY
DATA /XERXES,GREECE,PERSIA,TURKEY/10,5,4,3/
```

would define the record type XERXES to have the value 10, the set types GREECE, PERSIA, and TURKEY to have the values 5, 4, and 3, respectively. For Fortran Y, the above example would read

```
PARAMETER XERXES=10,GREECE=5,PERSIA=4,TURKEY=3
```

The programmer uses the name of the record or set type when issuing calls on the DBMS. For example, assuming that a record of type XERXES is the current record of the run unit, the following is valid

```
GET(PLATO,ATOSSA)
```

N.B. For PDP-11 Fortran, it is the programmer's responsibility NOT to assign any values to variables initialized for the programmer by the DBMS Fortran statements.

2.1.7.4 Miscellaneous Declarations

There are also certain identifiers defined to make some of the calls on the DBMS more symbolic and thus the intent of the call more apparent to any one looking at the application program listing. These identifiers are:

word	used for
ALL	arg in CNGSET, DELETE, GET, INSERT, MODIFY, REMOVE
ANY	dupctl in FINDK, OBTNK
AREA	currency update control
CURRNT	selector in FIND, OBTN
DBCLOS	argument in DBTERM or DBCLN
DBROLL	argument in DBTERM or DBCLN
ERL	error label indicator for Fortran-Y
ERR	error label indicator for PDP-11 Fortran
FIRST	selector in FIND, OBTN
LAST	selector in FIND, OBTN
NEXT	selector in FIND, OBTN
NULL	omitted record or set type in FIND, OBTN
NXTDUP	dupctl in FINDK, OBTNK
OWNER	selector in FIND, OBTN
PERM	selector in DELETE
PRIOR	selector in FIND, OBTN
RDONLY	read only indicator for DBOPEN
RECORD	currency update control
SCHEMA	alternate schema specifier
SELECT	selector in DELETE
SETS	currency update control
SIMPLE	selector in DELETE
UPDATE	modification declaration in currency-option list

The identifiers are used as any other identifier. For example,

```
GET(ALL)
```

The values of these identifiers are defined by Fortran statements identical to those used for defining record and set types.

N.B. For PDP-11 Fortran, it is the programmer's responsibility NOT to assign any values to variables initialized for the programmer by the DBMS Fortran statements.

2.1.7.5 Status Declarations and Error Processing

In order for the application program to access the DBMS status returns, a labeled common area called STATUS is defined in each application program. The statements provided are:

```
      INTEGER RECTYP,LUN,ERRLAB
      LOGICAL*1 ERROR,ERREC,ERSET,ERAREA,ERITEM,ERSTAT(4)
      EQUIVALENCE (ERROR,ERSTAT(1)),(ERREC,ERSTAT(2)),
     1   (ERSET,ERSTAT(3)),(ERAREA,ERSTAT(4)),(ERITEM,ERSTAT(4))
      COMMON/STATUS/FILLER,ERSTAT,ERRLAB,RECTYP,LUN
```

These statements are the PDP-11 Fortran version; the Fortran Y version would be slightly different. On the H6000, the labeled common name is <schema name>S and the error variables are integers.

After the completion of a DML procedure, if there was no error detected, the record type of the current record of the run unit will be stored in RECTYP.

After completing a DML procedure call, if there was an error, information can be obtained from the ERSTAT array by using the following names:

name	meaning
ERROR	error code
ERREC	record type of record associated with the error
ERSET	set type associated with the error
ERAREA	area type of area associated with the error
ERITEM	item number of item associated with the error

If one or more of the last three bytes in the array ERSTAT do not apply, the value will be zero. The use of the fourth byte, ERAREA or ERITEM, depends on the procedure call, that is, whether the procedure references specific fields in a record or not. If the ERITEM value is applicable for error processing, its value is the number of the item within the procedure call arguments. That is, the first item of the procedure call is number 1, the second number 2, and so forth.

At the completion of every DML procedure call, the DBMS tests to see if an error was detected during the execution of that procedure. If so, control is transferred to an error label. The error label can be specified to the DBMS in two ways, globally or locally. The global way is to call DBERR passing the error label. The local way is to specify the last two arguments of the DML procedure to be ERL followed by the value of the label. For example,

```
DBERR (ERL,$ELAB)
GET(ALL,ERL,$15)
```

where ELAB has been assigned the desired statement label.

A locally specified error label overrides a globally specified one for the procedure call in which it is specified. If neither a global nor a local error label is specified and an error occurs, the application program will be aborted and a message printed indicating the cause of the abort.

2.2 Status Codes

TABLE OF EXCEPTION CONDITIONS

Status Code	Data Base Exception Condition
	(Data Base Retrieval Exception Conditions)
021	End of set or area has been reached
023	No currency indicators established
024	No such record found
	(Currency Indicator Exception Conditions)
031	Current of area, record type, or set type is null
032	Current record of run unit is null
033	Current record of run unit is not correct record type
	(Data Base Exception Conditions)
041	Data base key is invalid
042	No access to record type or set type
044	No access to control item of sorted set
045	No write access to object record
	(Data Item Value Exception Conditions)
051	No duplicates allowed
054	Unknown item (no access to item)
055	No access to any items in record (record type may not contain any items)
056	Overflow on increment
	(Deleted Record Exception Conditions)
071	Access to a deleted record is specified
072	Deletion of a non-empty set is specified
073	Deletion of a unique record is specified
	(Membership Exception Conditions)
081	Object record is a member of set
082	Object record is a permanent member of set
083	Object record is not a member of set

TABLE OF EXCEPTION CONDITIONS

Status Code	Data Base Exception Condition
	(Resource Allocation Exception Conditions)
090	No more room to store object record
	(Miscellaneous Exception Conditions)
100	Set type or area name invalid
101	Record type invalid
102	Invalid selector specified
103	No sets found satisfying ALL request
104	Both record type and set type specified
105	Too many arguments specified
106	Too few arguments specified
107	Invalid item specification (item not declared as part of the record)
108	Bad record type or schema out of date (run unit schema working storage may be out of date)
109	Invalid currency update supression key word
111	Too many sets specified for currency update suppression
112	Bad schema pointer for DBOPEN
113	Data base not open
183	Too much PIUT space requested in DBOPEN
184	Too many user's running against the data base
185	Could not load DMLP

TABLE OF EXCEPTION CONDITIONS

status code	change	cngset	dbcln	dbopen	dbterm	delete	find	finda	findd	findk	get	incr	insert	memset	modify	movea	movec	mover	moves	movet	mtset	obtn	obtna	obtnd	obtnk	remove	store	stored
021							X	X														X	X					
023		X					X			X			X			X		X	X	X	X	X			X		X	X
024							X	X		X												X	X		X			
031		X					X	X					X									X	X				X	X
032	X	X				X					X	X	X	X	X											X		
033	X	X				X					X	X	X	X	X											X		
041									X															X			X	X
042		X					X	X	X	X			X	X							X	X	X	X	X	X	X	X
044										X															X		X	X
045	X	X				X	X	X	X	X		X	X		X							X	X	X	X	X	X	X
051		X											X		X												X	X
054	X										X	X			X													
055	X										X	X			X							X	X	X	X			
056												X																
071									X															X				
072						X																						
073						X																						
081													X															
082		X																								X		
083		X					X			X			X	X								X			X	X		

TABLE OF EXCEPTION CONDITIONS

status code	change	cngset	dbcln	dbopen	dbterm	delete	find	finda	findd	findk	get	incr	insert	memset	modify	movea	movec	mover	moves	movet	mtset	obtn	obtna	obtnd	obtnk	remove	store	stored
090																											X	X
100		X					X	X		X			X	X		X			X	X	X	X	X		X	X		
101							X	X		X								X				X	X		X		X	X
102						X	X	X														X	X					
103		X											X	X												X		
104							X	X														X	X					
105	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
106	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
107	X										X	X			X													
108							X	X		X	X				X			X				X	X		X		X	X
109							X	X	X	X												X	X	X	X		X	X
111							X	X	X	X												X	X	X	X		X	X
112				X																								
113	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
183				X																								
183				X																								
185				X																								

2.3 Prescan Service Subroutines

As one of its options, prescan will generate Fortran source code for any of three subroutines to load and dump designated records. These subroutines are:

Subroutine Name	Purpose
LOAD	Input data for record types
DUMP	Output data for record types in labeled form
RDUMP	Output data for record types in unlabeled form

The following three sections illustrate outputs from LOAD, DUMP, and RDUMP subroutines. These outputs are based on the following schema:

```
RECORD REC001
  ITEM CHAR01 IS 1 BYTE CHARACTER
  ITEM CHAR02 IS 5 BYTE CHARACTER
  ITEM CHAR03 IS 100 BYTE CHARACTER
  ITEM CHAR04 IS 3 BYTE CHARACTER OCCURS 7
  ITEM CHAR05 IS 100 BYTE CHARACTER OCCURS 2

RECORD REC002
  ITEM INT001 IS 2 BYTE INTEGER
  ITEM INT002 IS 2 BYTE INTEGER OCCURS 3

RECORD REC003
  ITEM REAL01 IS 4 BYTE REAL
  ITEM REAL02 IS 4 BYTE REAL OCCURS 3
  ITEM REAL03 IS 8 BYTE REAL
  ITEM REAL04 IS 8 BYTE REAL OCCURS 4
```

2.3.1 The Load Procedure

The LOAD procedure will input values for any or all data items of a specified record type from the user's terminal. Input values are written into the labeled common block corresponding to that record type. LOAD will solicit each data item, one at a time, by typing the item name, the index if the item is defined with an occurs clause, the item's type (C - character, I - integer, R - real) and the length for type character items. The following example shows the results of calls to LOAD on each of the three record types defined above. User responses are underscored.

```
LOAD RECORD REC001 (TYPE   1)
CHAR01       C(   1) A
CHAR02       C(   5) ABCDE
CHAR03       C( 100) ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMN
CHAR04(  1) C(   3) ABC
CHAR04(  2) C(   3) DEF
CHAR04(  3) C(   3) GHI
CHAR04(  4) C(   3) JKL
CHAR04(  5) C(   3) MNO
CHAR04(  6) C(   3) PQR
CHAR04(  7) C(   3) STU
CHAR05(  1) C( 100) AAAAAAAAAABBBBB
CHAR05(  2) C( 100) CCCCCCCCCCDDDDDDDDDDEEEEE

LOAD RECORD REC002 (TYPE   2)
INT001       I      1
INT002(  1)  I      2222
INT003(  2)  I      3333
INT002(  3)  I      4444

LOAD RECORD REC003 (TYPE   3)
REAL01       R      11111111.2222
REAL02(  1)  R      1111111111.
REAL02(  2)  R      4
REAL02(  3)  R      123456
REAL03       R      111.22
REAL04(  1)  R      1111111111.2222222222
REAL04(  2)  R      3
REAL04(  3)  R      4
REAL04(  4)  R      5
```

2.3.2 The Dump Procedure

The DUMP procedure prints out complete values for any or all data items of a specified record type. Data items printed by DUMP are obtained from the labeled common block associated with that record type. DUMP also prints the name of each item and its index in the case of an item occurring more than once. The following example shows the results of calls to DUMP on each of the three record types defined above.

```
RECORD REC001 (TYPE   1)
CHAR01       A
CHAR02       ABCDE
CHAR03       ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMN
CHAR04(  1) ABC
CHAR04(  2) DEF
CHAR04(  3) GHI
CHAR04(  4) JKL
CHAR04(  5) MNO
CHAR04(  6) PQR
CHAR04(  7) STU
CHAR05(  1) AAAAAAAAAABBBBB
CHAR05(  2) CCCCCCCCCCDDDDDDDDDDEEEEE

RECORD REC002 (TYPE   2)
INT001             1
INT002(  1)    2222
INT002(  2)    3333
INT002(  3)    4444

RECORD REC003 (TYPE   3)
REAL01        0.111111E 08
REAL02(  1)   0.111111E 10
REAL02(  2)   0.400000E-05
REAL02(  3)   0.123456
REAL03         111.220
REAL04(  1)   0.111111E 10
REAL04(  2)   0.300000E-05
REAL04(  3)   0.400000E-05
REAL04(  4)   0.500000E-05
```

2.3.3 The Record Dump Procedure

The RDUMP procedure prints out the first nine characters of any or all data items in the common block corresponding to a specified record type. RDUMP outputs have seven standard nine-character fields across the page and produce a very compact listing. Data items are not labeled. The following example shows the results of calls to RDUMP on each of the three records printed in the example of the previous section.

```
REC001 A         ABCDE     ABCDEFGHIABC       DEF       GHI       JKL
       MNO       PQR       STU       AAAAAAAAACCCCCCCCC

REC002           1      2222      3333      4444

REC003  0.11E 08 0.11E 10 0.40E-05 0.12      0.11E 03 0.11E 10
        0.30E-05 0.40E-05 0.50E-05
```

2.3.4 Subroutine Calling Sequence

All of the prescan service routines are called as follows:

```
CALL xxx(rec-name,items,n)
```

where

xxx is LOAD or DUMP or RDUMP

rec-name is a record type visible under subschema <access key>

items is an integer array containing the ordinals of the visible data tems in rec-name to be loaded or dumped. For example, to dump the first, third and fifth items of a record, the first three elements of the array should have the values 1, 3 and 5. Loading or dumping items which occur more than once will cause all occurrences to be processed.

n is the number of items to be loaded or dumped (number of entries in items). If n is greater than or equal to the number of visible data items in rec-name then all visible items will be transferred.

The following code would cause the first and third data items in the application program's subset of REC003 to be dumped to the terminal:

```
INTEGER ITEMS(2)
DATA ITEMS/1,3/

      .
      .
      .

CALL DUMP(REC003,ITEMS,2)
```

The following call would cause all items to be dumped:

```
CALL DUMP(REC003,ITEMS,4)
```

When the last argument is equal to or greater than the total number of items in the record, the contents of the second argument are not referenced. The following statement would also cause all items in REC003 to be dumped:

```
CALL DUMP(REC003,0,999)
```

3. DATA MANIPULATION LANGUAGE - Language Specifications

The Data Manipulation Language (DML) takes the form of a set of Fortran subroutine and function calls. Unless otherwise noted in the detailed description, the calls are for subroutines and so take the form of CALL x(y), where x is the subroutine name and y the sequence of arguments for the subroutine. If a DML call is specified to be a function, the call must be used where the function value can be saved or tested, such as an expression or assignment statement.

The description of the Fortran statements used to describe user record areas, item names, record type, set types and other names, the use of currency pointers, and other DML concepts are described in the previous section. Reference is made to the variable RECTYP, which is described above and contains the record type of the current record of the run unit.

If the values of an argument are specified in upper case then one of these words must be specified if the procedure is used. For example, NULL, SIMPLE, ALL. Other arguments, such as record-name, set-name are generic terms which must be replaced by the appropriate names, as described in the previous section.

3.1 The Change Procedure

CHANGE(item1, value1, item2, value2, ..., error-label)

The CHANGE procedure assigns specified values to up to four items of the current record of the run unit.

Rules:

1. The object record of the CHANGE procedure is the current record of the run unit. If the current record of the run unit is not of the type specified by the variable RECTYP, an error condition will occur.

2. The items must be defined as an integer item or character item of width 1 through 4 bytes within object record and these items may not be control items (e.g., calc or sort items) of that record.

3. If the items are defined as integer, the value will either be truncated or sign-extended if the width is not one word. If the items are defined as character, an error condition will occur if the item width is greater than 4.

4. Write access on a record type and access to the item is required to change an item in a record.

5. If more than one item is specified for changing and an error occurs, none of the items will be changed.

6. The error-label may be omitted, in which case the last label assigned to ERRLAB will be used in case an error occurs.

Comparison:

1. The CHANGE procedure is neither in CODASYL DBTG nor in IDS.

Status Codes:

032	Current record of run unit is null
033	Current record of run unit is not of correct type
045	No write access to object record
054	Unknown item (no access to item)
055	No access to any items in record
105	Too many arguments specified (more than 4 item-value pairs specified)
106	Too few arguments specified (the number of arguments may not be even)
107	Invalid item specification (item not declared as part of the record) (item is a control item or a floating point item)
113	Data base not open

Examples:

```
CALL CHANGE(SAM,10)
CALL CHANGE(PLATO,VAL,ATOSSA,0,ERR,IERR)
```

3.2 The Change Set Procedure

CNGSET(set-name1, set-name2, ..., error-label)

The CNGSET procedure performs set reassignment on the current record of the run unit. That is, for each set-name specified as an argument, CNGSET removes the record from the occurrence of the set type to which the record belongs and inserts it into the current occurrence of that set type.

Rules:

1. The object record of the CNGSET procedure is the current record of run unit. If the current record of the run unit is not of the type specified by the variable RECTYP, an error condition will occur.

2. The object record must be an optional or mandatory member of the specified sets.

3. For each set-name specified as an argument, the successful execution of CNGSET causes this record to be inserted into the current occurrence of set-name in accordance with the ordering criteria defined in the schema declaration for that set. The record becomes the current record of set-name.

4. The program must establish the desired occurrence of set-name in the currency indicators before the CNGSET procedure is executed.

5. Write access on a record type and access to the set is required to perform set reassignment.

6. If ALL is specified for set-names, set reassignment will occur for all sets to which the program has access and in which the object record is currently participating as an optional or mandatory member. Otherwise up to twelve sets may be specified for set reassignment.

7. If more that one set-name or ALL is specified and an error occurs then none of the set reassignments will be performed.

8. The error-label may be omitted, in which case the last label assigned to ERRLAB will be used in case an error occurs.

Comparison:

1. The CNGSET procedure is neither in CODASYL DBTG nor in IDS.

2. In CODASYL DBTG, set reassignment is performed as an option of the MODIFY procedure.

Status Codes:

023	No set currency indicators established
031	Current of set type is null
032	Current of run unit is null
033	Current record of run unit is not of correct type
042	No access to set type
045	No write access to object record
051	No duplicates allowed
082	Object record is a fixed record of set
083	Object record is not a member of set
100	Set type invalid
103	No sets found satisfying ALL request
105	Too many arguments specified (more than 12 set-names specified)
106	Too few arguments specified (no set-names specified)
113	Data base not open

Examples:

```
CALL CNGSET(XERXES,ERR,IERLAB)
CALL CNGSET(GREECE,TURKEY)
CALL CNGSET(ALL,ERR,IERR)
```

3.3 The Data Base Global Error Return Procedure

DBERR(error-label)

The DBERR procedure assigns an error label to the global error return ERRLAB. In MADMAN on the H6000 this procedure is the only method of assigning a global error return. Using ERS as the argument sets the global error label to the next statement following any call to a DML procedure that does not contain a specific error label.

Comparison

1. The DBERR procedure is neither in CODASYL DBTG nor in IDS.

Status Codes:

105 Too many arguments specified

Examples:

```
CALL DBERR(ERL,$LAB)
CALL DBERR(ERL,$10)
CALL DBERR(ERS)
```

3.4 The Data Base Open Procedure

```
DBOPEN(schema, subschema, piut-size, numb-buffs,
     ci-size, ro-indicator, error-label)
```

The DBOPEN procedure performs all necessary initialization to allow an application program to use a MADMAN data base.

Rules:

1. The schema argument is an identifier and specifies the name of the schema for the data base to be used. This must be the same name as used in the INVOKE statement.

2. The subschema argument is a key or key group name that specifies the subset of the data base to be used. This subset must be the same as used in the INVOKE statement. If no subsetting is used then this argument should be omitted or specified as zero.

3. The optional argument piut-size must be an integer and specifies an estimate of the number of pages expected to be used by the run unit. If this argument is omitted or specified as zero then the DBMS will use a default value. Since it is often difficult to estimate the exact number of pages expected to be accessed by a run unit the DBMS allows for some error. The DBMS will allow the run unit to access up to twice the number of pages specified by the piut-size argument before aborting the run unit for using too many pages. The default value and maximum allowable value for piut-size is controlled by the DBA using the data base utility.

4. The optional argument numb-buffs must be an integer and specifies the number of page buffers to be allocated for the run unit. If this argument is omitted or specified as zero then the DBMS will use a default value. The default value and maximum allowable value for numb-buffs is controlled by the DBA using the data base utility.

5. The optional argument ci-size must be an integer and specifies the maximum size for currency indicators. The value of the argument is in units of blocks with the size of 16 data base keys. If this argument is omitted or specified as zero then the DBMS will use a default value. During the execution of the run unit the currency indicators are allocated as they are established, for example, during a FIND or FINDK procedure. If the requested space is not sufficient then the run unit will be aborted and a message printed describing what happened. The default value and maximum allowable value for ci-size is controlled by the DBA using the data base utility.

6. The optional argument ro-indicator specifies RDONLY if the application program is an inquiry program. If RDONLY is

specified for this argument then the program will be aborted if it tries to update the data base.

7. The error-label may be omitted, in which case the last label assigned to ERRLAB will be used in case an error occurs.

Status Codes:

105	Too many arguments specified
106	Too few arguments specified
112	Bad schema pointer
183	Too much PIUT space requested
184	Too many user's running against the data base
185	Could not load the DMLP

Examples:

```
CALL DBOPEN(JUNKDB)
CALL DBOPEN(MASTER,KEYT1)
CALL DBOPEN(MASTER,KEYT2,150,4,5,ERR,IERR)
CALL DBOPEN(MASTER,DBAKEY,0,0,0,RDONLY)
```

3.5 The Data Base Terminate Procedure

```
DBTERM(indicator, error-label)
```

The DBTERM procedure terminates the availibility of the data base to the run unit. If the argument indicator specifies DBCLOS then any modifications made to the data base by the application program will become permanent. Otherwise, and in particular if the argument specifies DBROLL, all such modifications will be rolled back.

If the run unit terminates with the data base open then the DBMS will automatically roll back all modifications made to the data base by the program.

Status Codes:

105	Too many arguments specified
106	Too few arguments specified
113	Data base not open

Examples:

```
CALL DBTERM(DBROLL)
CALL DBTERM(DBCLOS)
```

3.6 The Data Base Cleanpoint Procedure

DBCLN(indicator, error-label)

The DBCLN procedure declares a clean point; that is, it operates as a simultaneous DBTERM-DBOPEN. All modifications made to the data base by the run unit will be made permanent or rolled back depending on whether the indicator specifies DBCLOS or DBROLL, respectively. The data base resources will be released and reinitialized. In particular, all currency indicators are cleared.

The application programmer should be aware that after a program executes a DBCLN procedure, values previously retrieved by the program may be changed by concurrent run units.

Status Codes:

105	Too many arguments specified
106	Too few arguments specified
113	Data base not open

Examples:

```
CALL DBCLN(DBCLOS)
CALL DBCLN(DBROLL)
```

3.7 The Delete Procedure

DELETE(selector, error-label)

The DELETE procedure deletes the current record of the run unit from the data base and, depending on the value of selector, possibly deletes additional records. The values of selector can be: SIMPLE, PERM, SELECT, or ALL.

Rules:

1. The object record occurrence of the DELETE procedure is the current record of the run unit. If the current record of the run unit is not of the type specified by the variable RECTYP, an error condition will occur.

2. If the selector specifies SIMPLE, the current record of the run unit must not be the owner of any set which contains any members, else an error condition will occur and no deletion will take place. [The function MTSET can be used to test for this condition.]

3. If the selector specifies PERM and the current record of the run unit is the owner of a set which contains members, then each OPTIONAL member will be removed from the set and each MANDATORY member will be deleted. The procedure is applied recursively to each deleted member.

4. If the selector specifies SELECT, it has an effect similar to PERM, except that an optional record will be deleted if it is found not to be a member in any sets.

5. If the selector specifies ALL and the current record of the run unit is the owner of a set with members, then all the members will be deleted. The procedure will be applied recursively to each deleted member.

6. Write access is required to delete a record from the data base. If the selector specifies PERM, SELECT, or ALL then the run unit must have write access to all possible record types that can be affected (deleted or removed) by the DELETE procedure.

7. Following the successful execution of the DELETE procedure, the current record of the run unit will be null. If the current of record type is one of the records that is deleted then that current of type will be made null. Currency indicators for sets which contained indicators for deleted or removed records participating as members will be made null; some of the other currency indicators (NEXT, PRIOR,...) may remain unchanged. Currency indicators for a set in which the owner was deleted will be made null.

8. The error-label may be omitted, in which case the last label assigned to ERRLAB will be used in case an error occurs.

Status Codes:

032	Current of run unit is null
033	Current of run unit is not of correct type
045	No write access to object record (may apply to any record type that can be affected)
072	Deletion of a non-empty set is specified
073	Deletion of a unique record is specified
102	Invalid selector specified
105	Too many arguments specified
106	Too few arguments specified
113	Data base not open

Examples:

```
CALL DELETE(SIMPLE)
CALL DELETE(ALL,ERR,IERR)
CALL DELETE(PERM)
```

3.8 The Find Procedure

FIND(selector, set-name, record-name, currency-option1, currency-option2, ..., error-label)

The FIND procedure locates and establishes record occurrence:

The current record of run unit;

The current record of record-name;

The current record of set for all sets-names in which the record currently participates as an owner or member;

The current record of the area in which the record is stored.

Some of the currency updating can be suppressed.

Rules:

1. The object record to be made current is specified by the selector expression of the argument list. The permissible values of selector are CURRNT, NEXT, PRIOR, FIRST, LAST, and OWNER. The following describes the action of FIND depending on the value of selector.

a.) CURRNT: Either the current record occurrence of the specified record-name or the current record occurrence of the specified set-name is selected as the object record depending upon which of the two arguments is given. The argument that is not used must be specified as NULL for this selector.

b.) NEXT, PRIOR, FIRST, LAST: The object record to be selected will be a member of the set specified by set-name. The occurrence of the set is identified by the currency indicators for that set type. Only record types to which the run unit has access will be considered in selecting the object record. The NEXT or PRIOR expression will locate the next or prior record of the specified set with respect to the logical order of the specified set. The FIRST or LAST expression will locate the first or last record with respect to the logical order of the specified set. If the record-name is also specified, only occurrences of that record type will be considered in selecting the object record. Otherwise, record-name must be specified as NULL.

c.) OWNER: The owner record of the current occurrence of the specified set-name will be selected as the object record. Since owners of singular sets are unique records,

specifying OWNER for a singular set is equivalent to specifying CURRNT for the owner record type.

2. The record selection expression determines whether the set-name or record-name argument items is needed. If one of these two arguments is not used, it must be specified as NULL.

3. The currency-option arguments are optional and need not be specified. They allow for programs to selectively control currency updating. The permissible values and meanings are:

currency-option	meaning
RECORD	suppress record currency updates
SETS	suppress all set currency updates
AREA	perform area currency updates (normally AREA currency updates are suppressed)
set-name1,set-name2,...	suppress currency updates for the specified sets (maximum of 5)

The currency options are independent except that if SETS and individual set-names are both specified, it will be taken to mean suppress all set currency updates except for the specified set-names. The arguments in the currency option list may occur in any order.

4. The word UPDATE may appear in the currency option list. If it does, it specifies that the run unit will modify the object record during its execution. The run unit should specify UPDATE only if it will definitely modify the record, since the UPDATE option makes the object record unavailable to other concurrent run units.

5. Currency updating will take place only for sets to which the run unit has access.

6. No actual movement of data is generated by the FIND procedure, only currency updating.

7. The error-label may be omitted, in which case the last label assigned to ERRLAB will be used in case an error occurs.

Status Codes:

021	End of set has been reached (for selector = NEXT or PRIOR)
023	No set currency indicators established
024	No such record found (for selector = FIRST or LAST)
031	Current of set type or record type is null
042	No access to set type or record type
045	No write access to object record (may occur if UPDATE option used)
055	No access to any item in record (may occur with OBTN)
083	Object record can not be a member of set
100	Set type invalid
101	Record type invalid
102	Invalid selector specified
104	Both record type and set type specified
105	Too many arguments specified
106	Too few arguments specified
108	Bad record type or schema out of date
109	Invalid currency update suppression key word
111	Too many sets specified for currency update suppression
113	Data base not open

Examples:

```
CALL FIND(CURRENT,NULL,SAM)
CALL FIND(CURRNT,DICK,NULL,RECORD,ERR,IERR)
CALL FIND(NEXT,DICK,SAM,SETS,UPDATE,ERL,$10)
CALL FIND(LAST,SALLY,NULL,AREA,SETS,SALLY,DICK)
CALL FIND(OWNER,PHIL,NULL,UPDATE,ERR,IERLAB)
```

3.9 The Find By Key Procedure

FINDK(record-name, dupctl, key-value, currency-option1, currency-option2, ..., error-label)

The FINDK procedure locates and establishes record occurrence based upon a specified key-value.

Currency updating and suppression are the same as for the FIND procedure.

Rules:

1. The object record to be made current must have been defined as a CALC record type or a member of a sorted (or indexed) set.

2. The permissible values for dupctl are ANY or NXTDUP or a set-name. ANY or NXTDUP are permissible only with records defined as CALC; set-name is permissible with records defined as members of sorted sets.

3. If ANY is specified, the first (or possibly only) record with the specified key-value is found. The NXTDUP value permits the user program to find additional records having the same key value. The search for the next duplicate starts from the last record with the identical key value. To find all the records with the same key value, use FINDK with ANY to find the first such record and then use a sequence of calls on FINDK with NXTDUP (and the same key value) to find the rest. Note that FINDK with NXTDUP will be treated as FINDK with ANY if the key values are different.

4. If dupctl specifies a set-name then the set will be searched to find the member with the specified key value. The proper set occurrence must have been established in the currency indicators.

5. FIND procedure rules 3-7 also cover the FINDK procedure.

Status Codes:

023	No set currency indicators established
024	No such record found
042	No access to set type or record type
044	No access to control item of sorted set
045	No write access to object record (may occur if UPDATE option used)
055	No access to any item in record (may occur with OBTNK)
083	Object record can not be a member of set
100	Set type invalid (set may not be sorted, sorted within type, or indexed)
101	Record type invalid
105	Too many arguments specified
106	Too few arguments specified
108	Bad record type or schema out of date
109	Invalid currency update suppression key word
111	Too many sets specified for currency update suppression
113	Data base not open

Examples:

```
CALL FINDK(SAM,ANY,KEY)
CALL FINDK(XERXES,NXTDUP,KEYA,AREA,SETS,SALLY,ERR,IERR)
CALL FINDK(XERXES,SALLY,KEYB,ERL,$100)
CALL FINDK(SARAH,DICK,NXTDUP,KEYB,ERR,IERR)
```

3.10 The Find Direct Procedure

```
FINDD(data-base-key, currency-option1,
    currency-option2, ..., error-label)
```

The FINDD procedure locates and establishes record occurrence directly based upon a specified data base key.

Currency updating and suppression are as defined for the FIND procedure.

Rules:

1. The data-base-key argument must evaluate to a variable whose value contains the desired data base key.

2. Any record to which the run unit has access may be directly located by the FINDD procedure.

3. FIND procedure rules 3-7 also apply to the FINDD procedure.

4. The user program may obtain the data base key of a record by using one of the MOVE procedures.

5. Data base keys should not be saved and used across executions of a run unit. The data base administrator reserves the right to restructure the data base at any time.

Status Codes:

041	Data base key is invalid
042	No access to record type
045	No write access to object record (may occur if UPDATE option used)
055	No access to any item in record (may occur with OBTND)
071	Access to a deleted record is specified
105	Too many arguments specified
106	Too few arguments specified
109	Invalid currency update suppression key word
111	Too many sets specified for currency update suppression
113	Data base not open

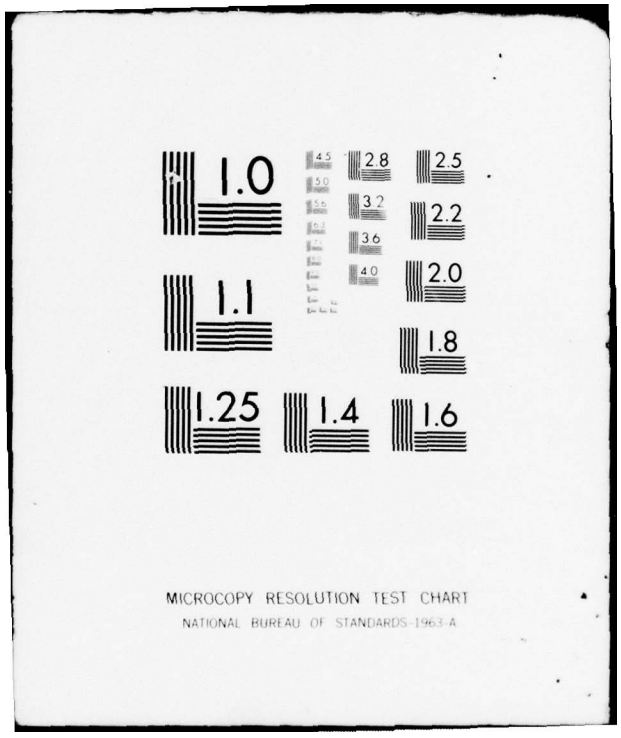
Examples:

```
CALL FINDD(DBK)
CALL FINDD(DBKA,AREA,RECORD,ERR,IERR)
```

AD-A056 376 GENERAL ELECTRIC CO SCHENECTADY N Y RESEARCH AND DEV--ETC F/G 9/2
MADMAN ON THE H6000: DATA BASE MANAGER FOR HONEYWELL 6000. VOLU--ETC(U)
JUN 78 D E PHILLIPS, J M BROWN, J P KELLERMAN F30602-76-C-0321
UNCLASSIFIED RADC-TR-78-8-VOL-2 NL

2 OF 2
AD
A056376

END
DATE
FILMED
9 -78
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

3.11 The Find By Area Procedure

FINDA(selector, area-id, record-name, currency-option1, currency-option2, ..., error-label)

The FINDA procedure locates and establishes record occurrences on an area basis. It functions like the FIND procedure except that an area-id is specified rather than a set-name.

Currency updating and suppression are similar to that for the FIND procedure.

Rules:

1. The object record to be made current is specified by the selector expression of the argument list. The permissible values of selector are CURRNT, NEXT, PRIOR, FIRST, LAST. The following describes the action of FINDA depending on the value of selector.

a.) CURRNT: The current record occurrence of the area is selected as the object record; record-name must be specified as NULL.

b.) NEXT, PRIOR, FIRST, LAST: The object record will be a member of the specified area. The logical ordering of the area is by data base key; NEXT meaning record with next highest data base key, PRIOR with next lowest. Only records to which the run unit has access will be considered in selecting the object record. The NEXT or PRIOR expression will locate the object record that is the next or prior record from the current of area. The FIRST or LAST expression will locate the first or last record with respect to the entire area. If the record-name is also specified, only occurrences of that record type will be considered in selecting the object record. Otherwise record-name must be specified as NULL.

2. FIND procedure rules 3-7 also apply to the FINDA procedure except that the meaning of AREA in the currency option list is reversed. That is, with FINDA, currency updating for the area will take place unless the word AREA appears in the currency option list.

Status Codes:

021	End of area has been reached (for selector = NEXT or PRIOR)
024	No such record found (for selector = FIRST or LAST)
031	Current of area is null
042	No access to record type
045	No write access to object record (may occur if UPDATE option used)
055	No access to any item in record (may occur with OBTNA)
100	Set type or area name invalid
101	Record type invalid
102	Invalid selector specified
104	Both record type and area name specified
105	Too many arguments specified
106	Too few arguments specified
108	Bad record type or schema out of date
109	Invalid currency update suppression key word
111	Too many sets specified for currency update suppression
113	Data base not open

Examples:

```
CALL FINDA(CURRNT,A1,NULL,AREA)
CALL FINDA(PRIOR,KINGDM,XERXES,SET12,SET123,SETS)
CALL FINDA(LAST,A17536,NULL,RECORD,ERR,IERR)
```

3.12 The Get Procedure

GET(item1, item2, ..., error-label)

The GET procedure causes one or more items of the current record of the run unit to be transferred to the user working storage area for a record of that type.

Rules:

1. The GET procedure operates on the current record of the run unit. If the current record of the run unit is not of the type specified by the variable RECTYP, an error condition will occur.

2. If ALL is specified for the item list, all parts of the record to which the program has access will be transferred to the user working storage. Otherwise up to ten items may be specified for transferral.

3. If item names are specified in the arguments, they must be the names of items to which the application program has access. Only those specified items will be moved to user working storage.

4. If the item name specifies a multiply occurring item then the argument will be taken to mean the first occurrence of that item. The only way all occurrences of such an item can be referenced without listing them individually is with the ALL specification.

5. The GET function can be obtained automatically following a successful FIND, FINDK, FINDD, or FINDA procedure by using an OBTN, OBTNK, OBTND, or OBTNA procedure.

6. The error-label may be omitted, in which case the last label assigned to ERRLAB will be used in case an error occurs.

Status Codes:

032	Current of run unit is null
033	Current of run unit is not of correct type
054	Unknown item (no access to item)
055	No access to any item in record
105	Too many arguments specified (more than 10 items specified)
106	Too few arguments specified (no items specified)
107	Invalid item specification (item not declared as part of the record)
108	Bad record type or schema out of date
113	Data base not open

Examples:

```
CALL GET(ALL)
CALL GET(PLATO,ATOSSA)
```

3.13 The Increment Procedure

INCR(item1, value1, item2, value2, ..., error-label)

The INCR procedure adds specified signed increments to up to four items of the current record of the run unit.

Rules:

1. The object record of the INCR procedure is the current record of the run unit. If the current record of the run unit is not of the type specified by the variable RECTYP, an error condition will occur.

2. The item must be defined as an integer item within the record in the schema definition and may not be a control item (e.g., calc or sort item) of that record.

3. The value is treated as a signed value, so that the specified item can be incremented or decremented. The width of the value is one word. If the declared width of item is less than a word, only the low order byte(s) will be used as the increment. If the declared width of item is greater than a word, the value will be sign-extended to the correct width for the addition.

4. Write access on a record and access to the item is required to increment an item in a record.

5. If more than one item is specified for incrementing and an error occurs, none of the items will be incremented.

6. The error-label may be omitted, in which case the last label assigned to ERRLAB will be used in case an error occurs.

Comparison:

1. The INCR procedure is neither in CODASYL DBTG nor in IDS.

Status Codes:

032	Current of run unit is null
033	Current of run unit is not of correct type
045	No write access to object record
054	Unknown item (no access to item)
055	No access to any item in record
056	Item overflowed on increment
105	Too many arguments specified (more than 4 item-value pairs specified)
106	Too few arguments specified (the number of arguments is not even)
107	Invalid item specification (item not declared as part of the record) (item is a control item or is not an integer item)
113	Data base not open

Examples:

```
CALL INCR(SAM,1)
CALL INCR(PLATO,-1,ATOSSA,-75,ERR,ERLBL)
```

3.14 The Insert Procedure

INSERT(set-name1, set-name2, ..., error-label)

The INSERT procedure causes the current record of run unit to become a member of the current occurrence of the specified sets.

Rules:

1. The object record occurrence of the INSERT procedure is the current record of the run unit. If the current record of the run unit is not of the type specified by the variable RECTYP, an error condition will occur.

2. The object record must be declared as a manual or an optional automatic member of set-name in the schema declaration, else an error is reported.

3. The object record is inserted into the current occurrence of set-name in accordance with the ordering criteria defined in the schema declaration for that set. The record becomes the current record of set-name.

4. The current record of the run unit must not be a member of any occurrence of set-name. Successful execution of INSERT will cause the specified record to become a member of an occurrence of set-name.

5. The user must establish the desired occurrence of set-name in the currency indicators before the INSERT procedure is executed.

6. Write access on the object record and access to the set is required to insert the record into a set.

7. If ALL is specified for the set-names, the record is inserted into all sets to which the application program has access and for which the record can participate as an manual or optional automatic member, but is not currently participating. Otherwise up to twelve sets may be specified for insertion.

8. If more than one set-name or ALL is specified and an error occurs then none of the insertions will be performed.

9. The error-label may be omitted, in which case the last label assigned to ERRLAB will be used in case an error occurs.

Status Codes:

023	No set currency indicators establish[illegible]
031	Current of set type is null
032	Current of run unit is null
033	Current of run unit is not of correct type
042	No access to record type
045	No write access to object record
051	No duplicates allowed
081	Object record is a member of set
083	Object record can not be a member of set
100	Set type invalid
103	No sets found satisfying ALL request
105	Too many arguments specified (more than 12 set-names specified)
106	Too few arguments specified (no set-names specified)
113	Data base not open

Examples:

```
CALL INSERT(SETABC)
CALL INSERT(ALL)
CALL INSERT(TURKEY,FRIED,ERR,IERR)
```

3.15 The Modify Procedure

MODIFY(item1, item2, ..., error-label)

The MODIFY procedure replaces one or more data items of the current record of the run unit, with the corresponding values from the user working area.

Rules:

1. The object record of the MODIFY procedure is the current record of the run unit. If the current record of the run unit is not of the type specified by the variable RECTYP, an error condition will occur.

2. The values of the appropriate data items for the record must be put in the record area in the user working storage. Only the specified items will be modified.

3. Items in the record to which the run unit does not have access will remain unchanged.

4. Control items such as CALC and SORT item values may be modified. A modified CALC item value will not physically move a record; however the record will still be properly located via the new CALC key value. A modified SORT item value will result in a logical reordering of the members of the set. The set occurrence is the one to which the object record currently belongs.

5. The values of the appropriate data items must be stored beforehand in the proper record area in the user working storage.

4. If the item name specifies a multiply occurring item then the argument will be taken to mean the first occurrence of that item. The only way all occurrences of such an item can be referenced without listing them individually is with the ALL specification.

7. If the item is specified as ALL then all items to which the program has access will be modified. In general, in such a case, the MODIFY procedure should be preceeded by a successful GET, STORE, or one of the obtain procedures to be sure all items in the user work area are initialized properly. If ALL is not specified then up to ten items may be specified for modification.

8. Write access on the record and access to the items is required to modify items of that record.

9. The error-label may be omitted, in which case the last label assigned to ERRLAB will be used in case an error occurs.

Status Codes:

032	Current of run unit is null
033	Current of run unit is not of correct type
045	No write access to object record
051	No duplicates allowed
054	Unknown item (no access to item)
055	No access to any item in record
105	Too many arguments specified (more than 10 items specified)
106	Too few arguments specified (no items specified)
107	Invalid item specification (item not declared as part of the record)
108	Bad record type or schema out of date
113	Data base not open

Examples:

```
CALL MODIFY(ALL)
CALL MODIFY(PLATO,ATOSSA,ERL,$LERRL)
```

3.16 The Move Current Of Run Unit Currency Procedure

```
MOVEC(identifier, error-label)
```

The MOVEC procedure causes the data base key for the current record of the run unit to be moved to the location specified by identifier within the program.

3.17 The Move Record Currency Procedure

```
MOVER(record-name1, identifier1,
     record-name2, identifier2, ..., error-label)
```

The MOVER procedure causes the data base key for the current record of record-name to be moved to the location specified by identifier within the user's program. The record-name--identifier pairs may be repeated up to four times and the data base keys will be moved to the indicated locations.

3.18 The Move Set Currency Procedure

```
MOVES(set-name1, identifier1,
     set-name2, identifier2, ..., error-label)
```

The MOVES procedure causes the data base key for the current record of set-name to be moved to the location specified by identifier within the user's program. The set-name--identifier pairs may be repeated up to four times and the data base keys of the specified set-names will be moved to the indicated locations.

3.19 The Move Set Table Currency Procedure

```
MOVET(set-name1, arrayname1,
     set-name2, arrayname2, ..., error-label)
```

The MOVET causes all the data base keys associated with the currency information for set-name to be moved to the specified array, within the user's program. Up to four moves can be specified with one procedure call. The data base keys are for the CURRENT, NEXT, PRIOR, and OWNER of set-name. Some of the entries may be null.

3.20 The Move Area Currency Procedure

```
MOVEA(area-id1, identifier1,
     area-id2, identifier2, ..., error-label)
```

The MOVEA procedure causes the data base key for the current record of area-id to be moved to the location specified by identifier, within the user's program. Up to four moves can be specified with one procedure call.

Notes:

1. The format of a data base key takes one word (four bytes). The first byte has the line number of the record, the next byte has the area number, and the last two bytes have the page number. Integer variables should be used for storing data base keys to prevent errors caused by floating point manipulations.

2. The error-label may be omitted, in which case the last label assigned to ERRLAB will be used in case an error occurs.

Status Codes:

023	No currency indicators established
100	Set type or area name invalid
101	Record type invalid
105	Too many arguments specified
106	Too few arguments specified
108	Bad record type or schema out of date
113	Data base not open

Examples:

```
CALL MOVEC(DBK)
CALL MOVER(SAM,DBK,XERXES,DBKX)
CALL MOVES(SETABC,SETDB1,SETCBC,SETDB2)
CALL MOVET(SAM,DBKTAB)
CALL MOVEA(KINGDM,DBK,ERR,IERR)
```

3.21 The Empty Set Function

MTSET(set-name, error-label)

MTSET is a logical valued function which is used to test whether the occurrence specified by the currency indicators for set-name is empty. Empty means that the set has no members, only an owner. The function value is true if the set is empty, false otherwise.

The error-label may be omitted, in which case the last label assigned to ERRLAB will be used in case an error occurs.

Status Codes:

023	No set currency indicators established
042	No access to set type
100	Set type invalid
105	Too many arguments specified
106	Too few arguments specified
113	Data base not open

Examples:

```
B=MTSET(SAM)
IF(MTSET(XERXES))FLAG=3.14159
```

3.22 The Membership Condition Function

MEMSET(set-name1, set-name2, ..., error-label)

MEMSET is a logical valued function which is used to test whether the current record of the run unit is a member of at least one occurrence of the specified set-names. Instead of specifying a list of set-names, ALL may be specified. In this case, the test is for all sets in which the record can participate and to which the run unit has access. If ALL is not specified then up to twelve sets can be specified for performing the membership test. The function value is true if the object record is a member of at least one of the specified sets, false otherwise.

The error-label may be omitted, in which case the last label assigned to ERRLAB will be used in case an error occurs.

Status Codes:

032	Current of run unit is null
033	Current of run unit is not of correct type
042	No access to set type
083	Object record can not be a member of set
100	Set type invalid
103	No sets found satisfying ALL request
105	Too many arguments specified (more than 12 set-names specified)
106	Too few arguments specified (no sets-names specified)
113	Data base not open

Examples:

```
IF(MEMSET(SAM))VALID=NO
BOOL=MEMSET(ALL)
```

3.23 The Obtain Procedure

OBTN(selector, set-name, record-name, currency-option1, currency-option2, ..., error-label)

The OBTN procedure locates and establishes a record occurrence and then moves the data items for the selected record to user working storage. It functions as a FIND followed by a GET with ALL as the argument.

The rules and status codes of the FIND procedure apply to this procedure, with the exception of rule 6.

3.24 The Obtain By Key Procedure

OBTNK(record-name, dupctl, key-value, currency-option1, currency-option2, ..., error-label)

The OBTNK procedure locates and establishes record occurrence based upon a specified key-value and then moves the data items for the selected record into user working storage. It functions as a FINDK followed by a GET with ALL specified as the argument.

The rules and status codes of the FINDK procedure apply to this procedure, except for rule 6 of the FIND procedure.

3.25 The Obtain Direct Procedure

```
OBTND(data-base-key, currency-option1,
    currency-option2, ..., error-label)
```

The OBTND procedure locates and establishes record occurrence directly based upon a specified data-base-key value and then moves the data items for the selected record into user working storage. It functions as a FINDD followed by a GET with ALL specified as the argument.

The rules and status codes of the FINDD procedure apply to this procedure, except for rule 6 of the FIND procedure.

3.26 The Obtain By Area Procedure

```
OBTNA(selector, area-id, record-name, currency-option1,
    currency-option2, ..., error-label)
```

The OBTNA procedure locates and establishes a record occurrence based on area membership and moves the data items for the selected record to user working storage. It functions as a FINDA followed by a GET with ALL specified as the argument.

The rules and status codes of the FINDA procedure apply to this procedure, except for rule 6 of the FIND procedure.

3.27 The Remove Procedure

REMOVE(set-name1, set-name2, ..., error-label)

The REMOVE procedure removes the current record of the run unit from the specified sets.

Rules:

1. The object record of the REMOVE procedure is the current record of run unit. If the current record of the run unit is not of the type specified by the variable RECTYP, an error condition will occur.

2. The object record must be declared as an optional member of set-name in the schema declaration, else an error is reported.

3. The object record must be a member of the specified set, else an error is reported. The successful execution of REMOVE will cancel that participation in set-name.

4. If the removed record is current of set-name, the current of set-name will become null. If it is in the currency table as PRIOR or NEXT, the entry will be updated to reflect the removal.

5. Write access on a record type is required to remove a record from a set.

6. If ALL is specified for the set-names, the record is removed from all sets to which the program has access and in which the record participates as an optional member. Otherwise up to twelve sets may be specified for removal.

7. If more than one set-name or ALL is specified and an error occurs then none of the removals will be performed.

8. The error-label may be omitted, in which case the last label assigned to ERRLAB will be used in case an error occurs.

Status Codes:

032	Current of run unit is null
033	Current of run unit is not of correct type
042	No access to set type
045	No write access to object record
082	Object record is a permanent member of set
083	Object record is not a member of set
100	Set type invalid
103	No sets found satisfying ALL request
105	Too many arguments specified (more than 12 set-names specified)
106	Too few arguments specified (no set-names specified)
113	Data base not open

Examples:

```
CALL REMOVE(SAM,XERXES)
CALL REMOVE(ALL,ERR,IERR)
```

3.28 The Store Procedure

```
STORE(record-name, currency-option1,
     currency-option2, ..., error-label)
```

The STORE procedure causes a new record to be placed in the data base.

Rules:

1. The values of the appropriate data items for the record must be put in the record area for record-name in the user working storage. The data items for all control data items such as CALC or sort items must be initialized.

2. Before the execution of the STORE procedure, the user must establish the current set occurrence for all sets in which the stored record will participate as an automatic member. The application program must have access to all sets in which the record participates as an automatic member.

3. The object record is established as the owner of a set occurrence for each set in which it has been defined as an owner. These set occurrences are empty at this time.

4. The successfully stored record becomes the current record of the run unit. The object record becomes the current record of its type and current of each set in which it participates as an owner or automatic member, unless certain of these currency updates are suppressed.

5. The location mode for the record type defined in the schema affects the physical placement of the new data record. The STORED procedure must be used to store records with a storage mode of direct.

6. The user must have write access on a record type to store it in the data base.

7. The currency-option arguments allow for user programs to selectively control currency updating. The permissible values and meanings are:

currency-option	meaning
RECORD	suppress record currency updates
SETS	suppress all set currency updates
AREA	perform area currency updates (normally AREA currency updates are suppressed)
set-name1,set-name2,...	suppress currency updates for the specified sets (maximum of 5)

The currency options are independent except that if SETS and individual set-names are both specified, it will be taken to

mean suppress all set currency updates except for the specified set-names. The arguments in the currency option list may occur in any order.

8. The error-label may be omitted, in which case the last label assigned to ERRLAB will be used in case an error occurs.

Status Codes:

023	No set currency indicators established
031	Current of set type is null
042	No access to set type or record type (possibly run unit does not have access to set in which record participates as an automatic member)
041	Data base key is invalid
044	No access to control item of sorted set
045	No write access to object record
051	No duplicates allowed
090	No more room
101	Record type invalid
105	Too many arguments specified
106	Too few arguments specified
108	Bad record type or schema out of date
109	Invalid currency update suppression key word
111	Too many sets specified for currency update suppression
113	Data base not open

Examples:

```
CALL STORE(SAM)
CALL STORE(XERXES,SETS,SALLY,ERR,IERR)
```

3.29 The Store Direct Procedure

```
STORED(dbk, record-name, currency-option1,
    currency-option2, ..., error-label)
```

The STORED procedure causes a new record to be placed in the data base. STORED is used to store records which have a storage mode of direct. The first argument specifies a location which contains a data base key. This data base key specifies the area number and the page number where the record is to be stored. The DBMS will assign the line number of the new record.

The rules and status codes for the STORE procedure apply to this procedure.

If the STORED procedure is used for record types which do not have a direct storage mode then the data base key is ignored.

Examples:

```
CALL STORED(DBK,JOHN)
CALL STORED(DBKA,PHIL,SETS,ERR,IERR)
```

MULTI ACCESS DATA BASE MANAGEMENT SYSTEM

(MADMAN FOR THE H6000)

DATA BASE UTILITY MANUAL

GENERAL ELECTRIC COMPANY

RESEARCH AND DEVELOPMENT CENTER

JUNE 1977

1. INTRODUCTION

The data base utility functions are performed by two programs: Data Base Utility One (DBU1) and Data Base Utility Two (DBU2).

Data Base Utility One performs those operations that are done off-line (i.e. DBU1 is not a user of the DBMS) and those operations that require privity.

Data Base Utility Two performs those operations that are done on-line (i.e. DBU2 is a user of the DBMS).

DBU1 is normally run only at start up time while DBU2 can be run at any time, usually when the system is in operation. Some operations can be done in either DBU1 or DBU2.

Both utility programs are normally run in batch mode; however, they can be run in pseudo-timesharing mode by doing a $DAC to the input and output files. (see Appendix B).

1.1 The Parameter File

The parameter file is used by both utility programs. This file consists of a record for each data base. Each record contains the permanent parameters being used for a particular data base. The Data Base Administrator may have several parameter files if he so desires. He only need inform the utility program that he wishes to use another file through the PF command.

The parameter file must be created by using the ACCESS subsystem before any data base whose parameters it will contain can be put on-line. (See Appendix C.)

1.2 Notation

ø = blank

[ ] = optional command argument

< > = required command argument

2. DATA BASE UTILITY ONE

2.1 Available functions

1. Change parameter file name (PF)
2. Permanent parameter change (PC)
3. Initialize data base (IN)
4. Verify data base (VE)
5. Start up data base (SU)
6. Terminate DBU1 (TE)

2.1.1 Change Parameter File Name Command

The PF command is used to specify a data base parameter file different from the default parameter file (UMC/DBPARMF). While the use of this command is optional, if used, it must appear as the first command in the DBU1 command stream.

FORMAT
PFØ[UMC]/<FILE NAME>[$PASSWORD]

Where:
UMC = User Master Catalog
FILE NAME = Parameter File Name
$PASSWORD = File Name Password

Note:
UMC is required only if the file is a member of a UMC different from the data base administrator UMC. Passwords are required only if there are passwords associated with the file. If passwords are needed, the currency sign ($) is a necessary delimiter.

2.1.2 Permanent Parameter Change Command

The PC command is used to permanently change the data base parameters.

The parameters which may be changed are listed below in the order in which they are expected in the PC command.

1. maximum number of page buffers
2. default number of page buffers
3. maximum currency indicator size parameter
4. default currency indicator size parameter
5. maximum page-in-use-table size parameter
6. default page-in-use-table size parameter

FORMAT:
PCØ[p1],[p2],[p3],[p4],[p5],[p6]

Where:
p1-p6 are parameters 1 through 6 as described above

Note: At least one parameter must be specified or an error will result.

If the PC command is not used, DBUl and the Data Base Management System use the current parameters in the parameter file. If there are no current parameters, the following values will be used:

maximum number of page buffers	4
default number of page buffers	4
maximum ci size parameter	10
default ci size parameter	5
maximum piut size parameter	100
default piut size parameter	50

The absolute maximum values for the parameters are:

maximum number of page buffers	4
default number of page buffers	*
maximum ci size parameter	46
default ci size parameter	*
maximum piut size parameter	128
default piut size parameter	*

Note: Those entries which have no value are set to the current value of the corresponding maximum parameter. The default ci size parameter has an additional constraint: it may not be larger than 16.

2.1.3 Initialize Command

The IN command causes DBU1 to initialize all of the data base files including the before file. DBU1 writes the skeleton for each page creating and growing files where necessary. Unique records are stored and a report indicating their locations is produced.

FORMAT:
IN

2.1.4 Verify Data Base Command

The VE command performs verification operations on the data base which includes verifying checksums, and verifying page identification. Errors are listed in a report produced by DBU1. If more than 10 errors occur, DBU1 is aborted and a cold restart (i.e. recovering the data base files from a previous system save) is required. VE assumes that the data base files exist and contain data base information. Therefore a new data base cannot be verified before it is initialized via the IN command. Of course any existing data base may be verified.

FORMAT:
VE

3. USING PRESCAN

PRESCAN is implemented in Fortran-Y. It can be run under both time-sharing and batch. As soon as PRESCAN begins execution it will request parameters from the user. The user enters the information from the terminal if he is in time-sharing or includes the information on a data card if he is in the batch mode. The user must specify an input file containing the application source code and an output file to which PRESCAN will copy the application program, adding data base declaration statements. Several options may also be specified. After processing each program, PRESCAN will request more parameters from the user. When running under time-sharing, PRESCAN may be terminated at this point by entering carriage return. When running under batch, PRESCAN is automatically terminated at end of input data.

3.1 PRESCAN Parameters

PRESCAN parameters are entered in the following format:

<input file>,<output file>[(<opt-1>,<opt-2>,...,<opt-n>)]

3.1.1 File formats

Input files can be line-numbered or not as the user desires. The line numbers can have any number of digits. However, they must be the first non-blank characters of the line. A file is taken to be line-numbered if the first line of the file starts with a valid line number.

File names cannot be longer than eight characters.

Input and output file names cannot be identical.

To specify input and output files:

Under time-sharing: the format is

[/cat/]filename

where cat is the subcatalog the user working under and filename is the name of the file.

Under batch: the format is

User-id/[cat/]filename

where user-id is the user's identification and cat and filename are as described above.

3. DATA BASE UTILITY TWO

3.1 Available Functions

1. Help (HE)
2. Change parameter file name (PF)
3. List data bases on-line (LN)
4. Normal shutdown (SD)
5. Emergengy shut down (TF)
6. List SNUMBS active against a data base (LS)
7. Abort a SNUMB (AS)
8. List permanent parameters (PL)
9. List parameters being used by an on-line data base (LP)
10. Permanent change of parameters (PC)
11. Change parameters of an on-line data base (CP)
12. Insert a schema record in the parameter file (IS)
13. Delete a schema from the parameter file (DS)
14. List data base statistics (ST)
15. Terminate DBU2 (TE)

3.1.1 HELP Command

The HELP command provides the user with command formats for all the DBU2 commands if "HELP" is typed as a command. Alternatively, the user can obtain the format of a specific DBU2 command by typing:

"HEØ<command mneumonic>".

FORMATS:

HELP	(for all commands)
HEØ<command mneumonic>	(for a specific command)

3.1.2 Change Parameter File Name Command

The PF command for DBU2 is similar to the PF command for DBU1 (section 2.1.1) except that it may occur at any point where the user wishes to work with a different parameter file.

3.1.3 List Data Bases On-line Command

The LN command lists all data bases on-line. To do this, DBU2 searches the parameter file for all active data bases and verifies that they are, in fact, on-line.

FORMAT:
LN

3.1.4 Normal Data Base Shut Down Command

The SD command causes the DBMS for the specified schema to refuse to accept new transactions but to allow all transactions running against it to terminate. The DBMS shuts itself down when all transactions have terminated.

FORMAT:
SDØ<schema>

3.1.5 Emergency Shut Down Command

The emergency shut down command (TF) is used to immediately shut down a data base. Transactions running against the data base are aborted, and their changes are rolled back. The DBMS is taken off-line.

FORMAT:
TFØ<schema>

3.1.6 List SNUMBS Active Against A Data Base Command

The LS command lists the SNUMBS of all transactions running against the specified data base.

FORMAT:
LSØ<schema>

3.1.7 Abort A SNUMB Command

This command will cause the DBMS to abort a specified transaction and roll back the changes.

FORMAT:
ASØ<schema>Ø<snumb>,<activity number>

3.1.8 List Parameters Command

This command lists the permanent parameters of a specified data base or, if no data base is specified, the permanent parameters for all data bases having a record in the current parameter file will be listed.

FORMAT:
PLØ[schema]

3.1.9 List Parameters Of On-line DBMS Command

The LP command lists the parameters being used by the specified data base.

FORMAT:
LPØ<schema>

3.1.10 Change Parameters Command

This command is used to change the parameters of a on-line data base. See section 2.1.2 for the order of the parameters.

FORMAT:
CPØ<schema>Ø[p1],[p2],[p3],[p4],[p5],[p6]

3.1.11 Permanent Parameter Change Command

The PC command for DBU2 is similar to the one for DBU1 (section 2.1.2) with several variations:

1. Since DBU2 can deal with more than one data base, the schema name must be specified.
2. There are no restrictions as to where the PC command may occur in the command stream of DBU2.
3. If no parameters are specified, the standard default parameters are used.
4. If the data base is on-line, a "change parameters" is also performed.

FORMAT:
PCØ<schema>Ø[p1],[p2],[p3],[p4],[p5],[p6]

3.1.12 Insert Schema Name Command

The IS command inserts a record into the parameter file with the specified schema name. If parameters are specified, they will be used as permanent parameters. If no parameters are specified, the standard default parameters will become the permanent parameters for that schema.

FORMAT:
IS∅<schema>∅[p1],[p2],[p3],[p4],[p5],[p6]

3.1.13 Delete Schema Command

This command removes the record for the specified schema from the parameter file and then repacks the parameter file.

FORMAT
DS∅<schema>

3.1.14 List Data Base Statistics Command

This command lists the following statistics of the specified data base.

1. number of active AP(s)
2. number of blocked AP(s)
3. maximum number of concurrent users
4. total number of conflicts
5. total number of blocks that have occurred
6. total number of roll backs
7. total number of pages read
8. total number of pages written
9. total number of pages rolled back
10. total number of aborts due to deadlock
11. total number of aborts due to all blocked
12. total number of transactions run

FORMAT:
ST∅<schema>

3.1.15 Terminate Command

The terminate command terminates execution of DBU2.

FORMAT:
TE

4. APPENDIX A - DBU1 - ERROR CODES

CODE NUMBER	SEVERITY	MEANING
1	<nf>	command unknown
2	<f>	bad parameter file name
3	<nf>	no parameters specified in PC command
4	<nf>	parameter much too large
5	<f>	schema name not in parameter file
6	<nf>	automatic recovery invoked
7	<nf>	on/off word in schema record is on
8	<f>	unrecoverable data base error-cold restart required
9	<f>	too many data base errors-cold restart
10	<nf>	not all data base files are open
11	<nf>	bad status on data base file open
12	<f>	area table consistency error
13	<f>	page size larger than model buffer
14	<f>	bad page size spec
15	<f>	unable to open data base
16	<f>	store unique error
17	<f>	movec error
18	<nf>	unable to grow file
19	<f>	file too small
20	<f>	file exists but cannot be opened
21	<f>	unable to create file
22	<f>	unable to allocate file just created
23	<nf>	file query error
24	<f>	read request returned a bad status
25	<f>	write request returned a bad status
26	<f>	unable to open parameter file (check to see that the file exists)
27	<f>	error occurred when reading parameter file

<f> is a fatal error.
<nf> is a non-fatal error.

5. APPENDIX B - CONTROL CARDS - DBU1 and DBU2

5.1 Control Cards for DBU1

To make a C* file of DBU1 use the following runstream:

```
$       IDENT   <userid>,*DBU1*
$       GMAP    DECK
$       SELECTA <userid>/DBU/DBU1MAC
$       SELECTA <userid>/DBU/DBU1DOCU
$       SELECTA <userid>/DBU/DBU1CON
$       SELECTA <userid>/DBU/DBU1
$       SELECTA <userid>/DBU/DBU1SUBS
$       SELECTA <userid>/DBU/DBU1INIT
$       SELECTA <userid>/DBU/DBU1REVE
$       SELECTA <userid>/DBU/DBU1STG
$       FILE    C*,X1R,10L,NEW,CSDBU1
$       ENDJOB
```

This will create a C* file named CSDBU1

5.2 Control Cards for DBU2

To compile DBU2 use the following runstream:

```
$       IDENT   <userid>,MCDBU2
$       OPTION  FORTRAN
$       FORTY   DECK,NFORM,NLNO
$       FILE    C*,X1R,1L,NEW,CSDBU2
$       SELECTA <userid>/DBU/DBU2S
$       ENDJOB
```

This will create the following C* files:

CSDBU2
CSGEINFO
CSDBU.I

To run DBU2 use one of the two following runstreams:

Running pseudo-timesharing ($DAC)

```
$       IDENT   <userid>,<banner>
$       OPTION  FORTRAN
$       USE     .RTYP
$       USE     .GTLIT
$       SELECT  <userid>/CSDBU2
$       SELECT  <userid>/CSGEINFO
$       SELECT  <userid>/CSDBU.I
$       SELECT  <userid>/CSIOFLI
$       SELECT  <userid>/CSFLIERR
$       ENTRY   DBU2
$       EXECUTE
$       DAC     05
$       DAC     06
$       ENDJOB
```

Running in batch mode

```
$       IDENT   <userid>,<banner>
$       OPTION  FORTRAN
$       USE     .GTLIT
$       SELECT  <userid>/CSDBU2
$       SELECT  <userid>/CSGEINFO
$       SELECT  <userid>/CSDBU.I
$       SELECT  <userid>/CSIOFLI
$       SELECT  <userid>/CSFLIERR
$       ENTRY   DBU2
$       EXECUTE
$       DATA    05
<DBU2 commands>
$       SYSOUT  06
$       ENDJOB
```

6. APPENDIX C - THE PARAMETER FILE

6.1 Format

The parameter file is a random file with a record size of 32 words. The first record in the file (record 0) is used for control information. Word 0 contains the number of records in the file. Records 1 through n (where n is the number of records in the file) contain the parameters for each Data Manager System using the file. The format of these records is as follows:

```
word 0     schema name (BCD)
word 1     maximum number of page buffers
word 2     default number of page buffers
word 3     maximum CI size
word 4     default CI size
word 5     maximum PIUT size
word 6     default PIUT size
word 7     on-line/off-line indicator (1=on, 0=off)
words 8-31 not currently used
```

6.2 Initialization

The parameter file must be created using the ACCESS subsystem using the following command:

```
ACCE CF,/<filename>,B/5,5/,MODE/R/,R,W,CLEAR
```

The default name used by both utility programs is
<userid>/DBPARMF

MULTI ACCESS DATA BASE MANAGEMENT SYSTEM

(MADMAN FOR THE PDP-11 AND THE H6000)

PRESCAN MANUAL

GENERAL ELECTRIC COMPANY

RESEARCH AND DEVELOPMENT CENTER

JUNE 1977

1. INTRODUCTION

DBMS application programs contain references to names of data base structures such as names of record types, data items, sets and areas. To compile correctly, applications programs must contain declaration statements defining these names. Since the application programmer ordinarily does not have sufficient information about the structure of the data base to code these statements, the PRESCAN preprocessor is used to generate and insert them into the application program prior to compilation. The applications programmer tells PRESCAN where to insert these declaration statements by placing a pseudo statement (INVOKE) in the source code.

For example, consider the following simple schema defining a data base with two record types and one set:

```
SCHEMA DEMO VERSION 1 DDL 1

RECORD REC1
  STORE UNIQUE IN AREA A1 ON PAGE 1
  ITEM INT1 IS 2 BYTE INTEGER
  ITEM INT2 IS 2 BYTE INTEGER OCCURS 5

RECORD REC2
  STORE PER SET1
  ITEM STR1 IS 8 BYTE CHARACTER
  ITEM STR2 IS 4 BYTE CHARACTER OCCURS 10
  ITEM REAL1 IS 4 BYTE REAL

SET SET1
  OWNER IS REC1
  INSERT AFTER
  MEMBER IS REC2 AUTOMATIC
```

If an applications program invoked this entire schema, the following declarations would be generated.

```
C
C SYMBOLIC AREA TYPES
C
      INTEGER A1
C
C SYMBOLIC RECORD TYPES
C
      INTEGER REC1,REC2
C
C SYMBOLIC SET TYPES
C
      INTEGER SET1
C
C RECORD TYPE REC1
C
      INTEGER INT1,INT2(5)
      COMMON/REC1/INT1,INT2
C
C RECORD TYPE REC2
C
      CHARACTER  STR1*8,STR2*4(10)
      REAL REAL1
      COMMON/REC2/STR1,STR2,REAL1
C
C DATA VALUES FOR AREA TYPES
C
      DATA A1/1/
C
C DATA VALUES FOR RECORD TYPES
C
      DATA REC1,REC2/1,2/
C
C DATA VALUES FOR SET TYPES
C
      DATA SET1/1/
```

These statements define all schema-dependent references to the data base. The labeled COMMON blocks defined for each record type can also be addressed by the Data Base Manager. All data transfers to and from the data base are made via these COMMON blocks. The GET procedure moves data from the data base to labeled COMMON. STORE moves data from labeled COMMON to the data base.

PRESCAN also generates declaration statements defining the following system variables and symbolic arguments: ERROR, ERREC, ERSET, ERAREA, ERITEM, ERSTAT, ERRLAB, RECTYP, LUN, SIMPLE, SELECT, PERM, ALL, CURRNT, FIRST, LAST, NEXT, PRIOR, OWNER, SETS, RECORD, AREA, NXTDUP, ANY, NULL, ERL, ERR, ERS, SCHEMA, DBROLL, DBCLOS, RDONLY and UPDATE.

A data base is usually divided into several subsets. This is accomplished by assigning access locks to all record types and data items in the data base. A number of access keys are then defined in such a way that one key may fit several different locks but not all locks. The data base subset which is to be referenced by a particular application program is designated by an access key in the INVOKE statement. PRESCAN will generate declaration statements for only those record types and data items whose access locks are matched by the access key in the INVOKE statement.

PRESCAN will optionally generate Fortran source code for subroutines that load and dump record images.

2. THE INVOKE STATEMENT

The format of the INVOKE statement is as follows:

INVOKE [SUBSCHEMA <access key> OF] SCHEMA <schema name>

PRESCAN will search the current working directory for a file named:

<schema name>.F

which contains a description of the entire schema including a list of legitimate access keys. <schema name>.F is created by the DDL compiler and has the same file format as described in Section 3.1.1. <access key>, if specified, must be a legitimate access key or group name defined in <schema name>. PRESCAN will replace the INVOKE statement with all record, set and data item declarations for <schema name> for which the associated access lock is matched by <access key>.

If <access key> is omitted and <schema name> contains no access section, all records, sets and data items from <schema name> will be declared. If <access key> is omitted but <schema name> does contain an access section, no declarations will be included in the application program.

The PDP-11 Fortran compilers require that DATA statements follow all other declaration statements. Therefore the INVOKE statement must follow the last declaration statement in the application program and precede the first DATA statement. For the H6000 FORTRAN-Y compiler, the INVOKE statement can appear anywhere before the first executable statement. "INVOKE" may begin in any column position but must be the first non-blank symbol on the line, not including the line number.

3. USING PRESCAN

PRESCAN is implemented in Fortran-Y. It can be run under both time-sharing and batch. As soon as PRESCAN begins execution it will request parameters from the user. The user enters the information from the terminal if he is in time-sharing or includes the information on a data card if he is in the batch mode. The user must specify an input file containing the application source code and an output file to which PRESCAN will copy the application program, adding data base declaration statements. Several options may also be specified. After processing each program, PRESCAN will request more parameters from the user. When running under time-sharing, PRESCAN may be terminated at this point by entering carriage return. When running under batch, PRESCAN is automatically terminated at end of input data.

3.1 PRESCAN Parameters

PRESCAN parameters are entered in the following format:

<input file>,<output file>[(<opt-1>,<opt-2>,...,<opt-n>)]

3.1.1 File formats

Input files can be line-numbered or not as the user desires. The line numbers can have any number of digits. However, they must be the first non-blank characters of the line. A file is taken to be line-numbered if the first line of the file starts with a valid line number.

File names cannot be longer than eight characters.

Input and output file names cannot be identical.

To specify input and output files:

Under time-sharing: the format is

[/cat/]filename

where cat is the subcatalog the user working under and filename is the name of the file.

Under batch: the format is

User-id/[cat/]filename

where user-id is the user's identification and cat and filename are as described above.

3.1.2 PRESCAN options

3.1.2.1 Options for Fortran formats

FOR - Generate all output files in PDP-11 FOR Fortran syntax.

F4P - Generate all output files in PDP-11 F4P Fortran syntax.

F/YXC - Generate all output files in H6000 Fortran-Y PDP-11 Cross Compiler Fortran syntax.

F/Y - Generate all output files in Fortran-Y syntax.

The user can choose one of the above three options.

3.1.2.2 Line numbering options

NLNO - output files will not be line numbered.

LNO,LNO5 - output files will have 5-digit line numbers.

The user can choose one of the above two options.

3.1.2.3 Format of output

FORM - output files will be formatted.

NFORM - output files will be unformatted.

The user can choose one of the above two options.

3.1.2.4 Options for service routines

LOAD - Generate a generalized record input subroutine for all records and data items visible under subschema <access key>.

DUMP - Generate a generalized record dump subroutine (long version) for all records and data items visible under subschema <access key>.

RDUMP - Generate a generalized record dump subroutine (abbreviated version) for all records and data items visible under subschema <access key>.

The user can choose any or all of the above three options.

If no options are specified, output would be in F/Y format with line numbers and formatted.

Unless otherwise specified, default formats for:
FOR and F4P are NLNO and FORM.
F/Y and F/YXC are LNO5 and NFORM.

3.2 Parameter Files

When parameters are requested by PRESCAN, the name of a file containing a number of parameter records may optionally be entered as follows:

%<filename>

This is useful when processing a number of application programs in one sitting. <filename> has the same file format as described in 3.1.1 and it containes only parameter records as described in 3.1. Nested references to parameter files are not allowed. When all records in the parameter file are processed, PRESCAN will request more parameters.

3.3 Sample Run Streams

3.3.1 Under time-sharing

The following run stream illustrates the processing of the application program WTESTL. User responses are underscored.

```
SYSTEM ? YFOR
OLD OR NEW-NEW
READY
*RUN PRESCAN

ENTER FILE NAMES <INPUT,OUTPUT[(OPTIONS)]>
=/WTESTL,/OUTFILE(F/Y,LNO5,FORM)
=
*BYE
```

3.3.2 Under batch

PRESCAN is usually run in the time-sharing mode. A version designed to run in batch is available. Information is available upon request.

4. GENERATED SERVICE SUBROUTINES (not currently available)

PRESCAN will optionally generate Fortran source code for three types of subroutines to load and dump designated records. These subroutines are named as follows:

Option	Subroutine Name	Output File Name
LOAD	LOAD	<output file>.I
DUMP	DUMP	<output file>.D
RDUMP	RDUMP	<output file>.R

When requested, they will be written to a file whose name is constructed from the output file name, a period and a designated letter. The file format would be the same as what the user specified in his options list. When service routines are requested, <output file> cannot be longer than six characters.

The following three sections illustrate outputs from LOAD, DUMP and RDUMP subroutines. These outputs are based on the following schema:

```
RECORD REC001
  ITEM CHAR01 IS 1 BYTE CHARACTER
  ITEM CHAR02 IS 5 BYTE CHARACTER
  ITEM CHAR03 IS 100 BYTE CHARACTER
  ITEM CHAR04 IS 3 BYTE CHARACTER OCCURS 7
  ITEM CHAR05 IS 100 BYTE CHARACTER OCCURS 2

RECORD REC002
  ITEM INT001 IS 2 BYTE INTEGER
  ITEM INT002 IS 2 BYTE INTEGER OCCURS 3

RECORD REC003
  ITEM REAL01 IS 4 BYTE REAL
  ITEM REAL02 IS 4 BYTE REAL OCCURS 3
  ITEM REAL03 IS 8 BYTE REAL
  ITEM REAL04 IS 8 BYTE REAL OCCURS 4
```

4.1 Subroutine LOAD

Subroutine LOAD inputs values for any or all data items of a specified record type from the user's terminal. Input values are written into the labeled COMMON block corresponding to that record type. LOAD will solicit each data item, one at a time, by typing the item name, the index if the item is defined with an occurs clause, the item's type (C - character, I - integer, R - real) and the length for type character items. The following example shows the results of calls to LOAD on each of the three record types defined above. User responses are underscored.

```
     LOAD RECORD REC001 (TYPE   1)
     CHAR01       C(   1) A
     CHAR02       C(   5) ABCDE
     CHAR03       C( 100) ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMN
     CHAR04(  1) C(   3) ABC
     CHAR04(  2) C(   3) DEF
     CHAR04(  3) C(   3) GHI
     CHAR04(  4) C(   3) JKL
     CHAR04(  5) C(   3) MNO
     CHAR04(  6) C(   3) PQR
     CHAR04(  7) C(   3) STU
     CHAR05(  1) C( 100) AAAAAAAAAABBBBB
     CHAR05(  2) C( 100) CCCCCCCCCCDDDDDDDDDDEEEEE

LOAD RECORD REC002 (TYPE   2)
     INT001       I       1
     INT002(  1) I      2222
     INT003(  2) I      3333
     INT002(  3) I      4444

LOAD RECORD REC003 (TYPE   3)
     REAL01       R      11111111.2222
     REAL02(  1) R      1111111111.
     REAL02(  2) R      4
     REAL02(  3) R      123456
     REAL03       R      111.22
     REAL04(  1) R      1111111111.2222222222
     REAL04(  2) R      3
     REAL04(  3) R      4
     REAL04(  4) R      5
```

4.2 Subroutine DUMP

Subroutine DUMP prints out complete values for any or all data items of a specified record type. Data items printed by DUMP are obtained from the labeled COMMON block associated with that record type. DUMP also prints the name of each item and its index in the case of OCCURs items. The following example shows the results of calls to DUMP on each of the three record types defined above.

```
RECORD REC001 (TYPE   1)
CHAR01       A
CHAR02       ABCDE
CHAR03       ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMN
CHAR04(  1) ABC
CHAR04(  2) DEF
CHAR04(  3) GHI
CHAR04(  4) JKL
CHAR04(  5) MNO
CHAR04(  6) PQR
CHAR04(  7) STU
CHAR05(  1) AAAAAAAAAABBBBE
CHAR05(  2) CCCCCCCCCCDDDDDDDDDDEEEEE

RECORD REC002 (TYPE   2)
INT001            1
INT002(  1)   2222
INT002(  2) 3333
INT002(  3) 4444

RECORD REC003 (TYPE   3)
REAL01        0.111111E 08
REAL02(  1)   0.111111E 10
REAL02(  2)   0.400000E-05
REAL02(  3)   0.123456
REAL03         111.220
REAL04(  1)   0.111111E 10
REAL04(  2)   0.300000E-05
REAL04(  3)   0.400000E-05
REAL04(  4)   0.500000E-05
```

4.3 Subroutine RDUMP

Subroutine RDUMP prints out the first nine characters of any or all data items in the COMMON block corresponding to a specified record type. RDUMP outputs have seven standard nine-character fields across the page and produce a very compact listing. Data items are not labelled. The following example shows the results of calls to RDUMP on each of the three records printed in the example of Section 4.2.

```
REC001 A         ABCDE     ABCDEFGHIABC       DEF         GHI
       JKL       MNO       PQR          STU   AAAAAAAAACCCCCCCCC

REC002           1      2222      3333      4444

REC003  0.11E 08 0.11E 10 0.40E-05 0.12      0.11E 03
        0.11E-10 0.30E-05 0.40E-05 0.50E-05
```

4.4 Subroutine Calling Sequence

All service routines are called as follows:

```
CALL xxx(rec-name,items,n)
```

where

xxx = LOAD or DUMP or RDUMP

rec-name = a record type visible under subschema <access key>

items = an integer array containing the ordinals of the visible data items in rec-name to be loaded or dumped. For example, to dump the first, third and fifth items of a record, the first three elements of the array should have the values 1, 3 and 5

n = number of items to be loaded or dumped (number of entries in items). If n is greater than the number of visible data items in rec-name then all visible items will be transferred.

The following code would cause the first and third data items in the application program's subset of REC003 to be dumped to the terminal:

```
INTEGER ITEMS(2)
DATA ITEMS/1,3/

      .
      .
      .

CALL DUMP(REC003,ITEMS,2)
```

The following call would cause all items to be dumped:

```
CALL DUMP(REC003,ITEMS,4)
```

When the last argument is equal to or greater than the total number of items in the record, the contents of the second argument are not referenced. The following statement would also cause all items in REC003 to be dumped:

```
CALL DUMP(REC003,0,999)
```